

高等学校信息安全专业规划教材

计算机安全与保密

李辉　主编

清华大学出版社

21世纪高等学校信息安全专业规划教材

计算机安全与保密

李 辉 编著

清华大学出版社
北 京

内 容 简 介

信息安全是计算机、通信及相关专业的一个新兴热点。本书以作者十多年的教学经验为基础，结合计算机专业的特点，汇集了许多学生感兴趣的主题，力图浅显易懂、循序渐进，并重视实例和应用，重视点和面的结合。

为便于安排教学，本书的内容一共分为10章。第1章介绍一些常见的古典密码系统算法；第2章介绍包括DES、AES在内的对称密码系统和国内教材很少提及的加密与解密模式；第3～5章介绍公钥密码系统，其中，第3章的公钥密码系统基于大数因式分解问题，第4章的公钥密码系统基于离散对数问题，第5章的一些公钥密码系统则基于其他的完全NP问题；第6章介绍数字签名的原理；第7章介绍Hash函数以及基于Hash函数和对称密码算法的消息验证码技术；第8章介绍密钥管理和PKI技术；第9章以专题的形式介绍目前计算机安全应用领域的一些主流技术，包括口令安全、VPN等；第10章基于Java平台介绍密码学及安全的编程框架和技术。以上各章中，第1～7章偏重理论与原理；第8章和第9章偏重于应用；第10章偏重于编程。

本书覆盖较多的知识点，可以作为高等院校计算机系教材，同时，书中提供大量的数据实例，也可以作为教师、科研人员以及兴趣爱好者的参考书。

图书在版编目(CIP)数据

计算机安全与保密/李辉编著. —北京：清华大学出版社，2013. 2
21世纪高等学校信息安全专业规划教材
ISBN 978-7-302-29387-3

Ⅰ. ①计…　Ⅱ. ①李…　Ⅲ. ①计算机安全—高等学校—教材 ②电子计算机—密码术—高等学校—教材　Ⅳ. ①TP309

中国版本图书馆CIP数据核字(2012)第158621号

责任编辑： 魏江江　王冰飞
封面设计： 杨　兮
责任校对： 焦丽丽
责任印制：

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn，http://www.wqbook.com
地　　址： 北京清华大学学研大厦A座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载： http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185mm×260mm　**印　　张：** 17.5　**字　　数：** 426千字
版　　次： 2013年2月第1版　**印　　次：** 2013年2月第1次印刷
印　　数： 1～　000
定　　价： .00元

产品编号：040608-01

出版说明

由于网络应用越来越普及，信息化的社会已经呈现出越来越广阔的前景，可以肯定地说，在未来的社会中电子支付、电子银行、电子政务以及多方面的网络信息服务将深入到人类生活的方方面面。同时，随之面临的信息安全问题也日益突出，非法访问、信息窃取、甚至信息犯罪等恶意行为导致信息的严重不安全。信息安全问题已由原来的军事国防领域扩展到了整个社会，因此社会各界对信息安全人才有强烈的需求。

信息安全本科专业是2000年以来结合我国特色开设的新的本科专业，是计算机、通信、数学等领域的交叉学科，主要研究确保信息安全的科学和技术。自专业创办以来，各个高校在课程设置和教材研究上一直处于探索阶段。但各高校由于本身专业设置上来自于不同的学科，如计算机、通信和数学等，在课程设置上也没有统一的指导规范，在课程内容、深浅程度和课程衔接上，存在模糊不清、内容重叠、知识覆盖不全面等现象。因此，根据信息安全类专业知识体系所覆盖的知识点，系统地研究目前信息安全专业教学所涉及的核心技术的原理、实践及其应用，合理规划信息安全专业的核心课程，在此基础上提出适合我国信息安全专业教学和人才培养的核心课程的内容框架和知识体系，并在此基础上设计新的教学模式和教学方法，对进一步提高国内信息安全专业的教学水平和质量具有重要的意义。

为了进一步提高国内信息安全专业课程的教学水平和质量，培养适应社会经济发展需要的、兼具研究能力和工程能力的高质量专业技术人才。在教育部相关教学指导委员会专家的指导和建议下，清华大学出版社与国内多所重点大学共同对我国信息安全人才培养的课程框架和知识体系，以及实践教学内容进行了深入的研究，并在该基础上形成了“信息安全人才需求与专业知识体系、课程体系的研究”等研究报告。

本系列教材是在课程体系的研究基础上总结、完善而成，力求充分体现科学性、先进性、工程性，突出专业核心课程的教材，兼顾具有专业教学特点的相关基础课程教材，探索具有发展潜力的选修课程教材，满足高校多层次教学的需要。

本系列教材在规划过程中体现了如下一些基本组织原则和特点。

(1) 反映信息安全学科的发展和专业教育的改革，适应社会对信息安全人才的培养需求，教材内容坚持基本理论的扎实和清晰，反映基本理论和原理的综合应用，在其基础上强调工程实践环节，并及时反映教学体系的调整和教学内容的更新。

(2) 反映教学需要，促进教学发展。教材要适应多样化的教学需要，正确把握教学内容和课程体系的改革方向，在选择教材内容和编写体系时注意体现素质教育、创新能

力与实践能力的培养，为学生知识、能力、素质协调发展创造条件。

(3) 实施精品战略，突出重点。规划教材建设把重点放在专业核心(基础)课程的教材建设上；特别注意选择并安排一部分原来基础比较好的优秀教材或讲义修订再版，逐步形成精品教材；提倡并鼓励编写体现工程型和应用型的专业教学内容和课程体系改革成果的教材。

(4) 支持一纲多本，合理配套。专业核心课和相关基础课的教材要配套，同一门课程可以有多本具有各自内容特点的教材。处理好教材统一性与多样化，基本教材与辅助教材、教学参考书，文字教材与软件教材的关系，实现教材系列资源的配套。

(5) 依靠专家，择优落实。在制定教材规划时依靠各课程专家在调查研究本课程教材建设现状的基础上提出规划选题。在落实主编人选时，要引入竞争机制，通过申报、评审确定主编。书稿完成后认真实行审稿程序，确保出书质量。

繁荣教材出版事业，提高教材质量的关键是教师。建立一支高水平的、以老带新的教材编写队伍才能保证教材的编写质量，希望有志于教材建设的教师能够加入到我们的编写队伍中来。

21 世纪高等学校信息安全专业规划教材

联系人：魏江江 weijj@tup.tsinghua.edu.cn

前　　言

尽管从事“计算机安全与保密”课程的一线课堂教学已超过十年，但当我们重新审视这门课程的时候，仍感觉不是那么随心所欲。与“操作系统”、“数据结构”等计算机专业课程相比，“计算机安全与保密”课程总显得有些特别。

“计算机安全与保密”课程并不是自成体系，涉及许多其他的课程，如“抽象代数”、“操作系统”、“计算机网络”、“数据库理论与应用”、“信息论与编码”等，似乎很复杂、很庞大。与此形成反差的是，每年开课时总能吸引不少没有太多计算机专业基础的其他专业的同学，他们来到课堂上津津有味地旁听。

“计算机安全与保密”的基础实验之一“AES 加密”，对于计算机专业的同学来说，大多数都需要 3 天左右，这已经超过了为整个课程配的上机总时长。一些同学认为这个实验必须做，否则相关部分的课程就白上了；另一些同学则认为这个实验耗时太长，还是不做为好。

同学们对这门课的侧重点也存在较大分歧。一些同学认为应侧重于应用，如怎样攻击或防范攻击是很重要的；另一些同学却认为真正有趣的还是那些玄妙的加密、解密方案。

感兴趣的同学很多且分歧较大，造成了该课程教学中的一个困难，即难以找到一本合适的教材。从教学实践上来看，这门课理想的教材应该至少具备以下的特点：

（1）内容编排要合理。既能使同学们在计算机安全方面提高素养和能力，又能使没有太多基础的同学保持浓厚兴趣并积极参与实践。

（2）重视方法和思路胜过具体知识和手段。陆续有一些方案被破解或认为不那么安全，但解决安全问题的思路不会有大的变化。

（3）理论与实践并重。

在整理了以往的讲义和学生问答后，我们决定编写本教材。受限于水平和能力，无法将教材编得这么理想，但这确实是我们不断进步的方向。

关于本教材，还要做以下说明：

（1）为使本书尽可能地系统且全面，本书中个别部分参考了国外教材中的实例，但本书中绝大多数构造的实例来源于课堂教学或实践教学，实例中的数据也均通过编程得到。

（2）为了帮助读者深入研究，某些重要的算法（如椭圆曲线）提供了非常大的数据实例。读者不必对这些“大数”产生恐惧或反感，因为相关部分往往还有一些小数据的

实例。而且即便是跳过这些实例，也不会影响对全书的理解。

(3)“香农理论”虽然是一个重要的理论基础，但根据同学们的建议，移到了附录A中。跳过“香农理论”并不影响对全书的理解。

(4) 教师可根据总课时安排教学内容。课时较多时，可逐章讲授。课时不多的情况下，可结合课时要求及侧重点选择以下的组合：

- 第1章→第2章→第3章→第6章→第7章→第9章
- 第1章→第2章→第3章→第6章→第7章→第10章
- 第1章→第2章→第3章→第4章→第5章→第10章
- 第1章→第2章→第3章→第10章
- 第1章→第2章→第3章→第4章→第5章
- 第1章→第2章→第3章→第6章→第7章
- 第1章→第2章→第3章

最后，感谢过去几年给这门课提出各种意见的同学们，同学们的热情和求知精神给予我们编写教材的动力。还要感谢为本教材编写提供帮助的编辑、老师和研究生。特别感谢张小韬、张瑞霞、吕宏强等几位研究生，他们的录入和校对工作对本教材的编写帮助很大。

由于本教材涉及知识面较广，难度较大，不足之处在所难免。为便于以后教材的修订，恳请专家、教师及各类读者多提宝贵意见。

编　者
2012年9月

目　　录

引　　言

进入 21 世纪后，信息处理和通信业务呈现出高速增长的态势。人们的生活越来越依赖于计算机和通信，人们的工作也越来越多地借助计算机网络来进行。各类信息（如文本、多媒体）以二进制流的形式在网络上穿梭。如果不采取任何安全措施，敏感信息很容易被获取。而另一方面出于开放互联的考虑，大家又不可能也不愿意在隔离的环境里受限地使用计算机网络。

幸运的是数学、密码学理论和信息安全技术可以帮助解决计算机、网络和通信中的关键安全问题。

当人们谈及计算机安全时，会发现随着计算机的发展，有关计算机安全的事件日渐增多。例如，1939 年第二次世界大战爆发期间，正在为英国国家密码机构工作的图灵成功破译了德国军方使用的著名通信密码系统“Enigma”，如图 1 所示；1970 年，美国威斯康星大学的国防数学研究中心遭到炸弹袭击，计算机与积累了 20 多年的数据被炸毁；1982 年，只有 15 岁的理查德・斯伦塔（Richard Shrenta）针对苹果 Apple Ⅱ计算机编写了名为 Elk Cloner 的病毒，被认为是全球第一款个人计算机病毒；1996 年 2 月，Master Card 和 Visa 国际信用卡组织与技术合作伙伴 GTE、Netscape、IBM 等一批跨国公司共同开发了安全电子交易规范（SET）；1999 年 9 月 13 日，我国正式发布了标准 GB17859—1999“计算机信息系统安全保护等级划分准则”（Classified Criteria for Security Protection of Computer Information System）……



图 1　Enigma 系统

这些与计算机安全相关的事件，反映了计算机安全领域的主题。可以看到，计算机安全的研究内容包括计算机数据安全、计算机物理安全、计算机安全管理等，涉及的范围非常广泛。

本书在内容上不求面面俱到，主要侧重于介绍密码学基本原理和以密码学为基础的计算机系统安全；而关于计算机物理安全、计算机安全管理、计算机病毒等内容，则不在本书范围之内。

表 1 总结了计算机系统安全的一些重要主题。限于篇幅，标有 * 号的内容在本书中并没有详细介绍，感兴趣的读者可以查阅相关文献。

密码学在计算机系统安全领域中起着举足轻重的作用。一方面，在已有的与计算机系统安全相关的各类协议和标准中，绝大多数都以密码学为基础。另一方面，仍有不少计算机系统安全的遗留问题或新的应用需求需要借助密码学寻求解决方案。

表 1　计算机系统安全的主题

主　题	说　明
隐私或保密	仅让授权的实体得到信息，以保持信息的机密性
数据完整性	保证信息不被未授权的更改
实体认证或者身份鉴别	实体身份验证(如人、计算机终端、信用卡等)
消息认证	验证信息的来源，即原始数据认证
签名	一种把信息与实体绑定的手段
授权*	允许实体进行操作或成为某个角色
验证	提供授权实体在有效时间内使用信息或资源的一种手段
访问控制*	限制资源访问，资源只允许被授权的实体访问
证书	一个可信赖机构提供的证明文件
时间戳*	记录信息生成或者存在的时间
见证*	由创建者之外的实体证明信息的有效性
收据*	确认信息已经收到
证实*	确认服务已经提供
拥有*	让实体得到合法授权的一种手段
匿名*	在一些过程中隐藏实体的身份
不可抵赖性	避免否认先前的承诺或者行为
撤销*	一般指证书或授权的取消

与计算机的产生相比，密码学可谓历史悠久。本书中介绍的密码算法可以追溯到古罗马时期，而密码术思想的出现甚至还要更早。

根据《高卢战记》的记载，凯撒大帝常常在情报传输中采用一种神奇的加密方法：将书信中用到的所有字母按字母表顺序推后三位，信件完成后将信件卷起，然后将厚厚的蜡滴于封口处，在蜡没干之前，再压上自己的私印。接收者收到信件以后，先检查蜡封是否完整以及蜡印是否是凯撒的印章，然后拆开信件，将信中所有字母按字母表顺序提前三位，重新生成信件内容。

在凯撒密码中，原始的信息经加密后发出，加密前的信息称为明文(Plain Text)。加密规则(Encryption Rule)是字母顺序推后三位，如用 D 替代 A、用 F 替代 C。收到的信息无法理解，因为它是经过加密的信息，称为密文(Cipher Text)。要想还原出原来的明文信息，需要套用解密规则(Decryption Rule)，解密规则与加密规则正好相反，如用 A 替代 D、用 C 替代 F。解密后重要信息跃然纸上，表 2 中经解密后的明文拉丁文意思是“高卢全境分为 3 个部分”。

表 2　凯撒密码实例

收到的信息	R	p	q	l	d	J	d	o	o	l	d	h	v	w	G	l
还原的信息	O	m	n	i	a	G	a	l	l	i	a	e	s	t	D	i
收到的信息	y	l	v	d	l	q	S	d	u	w	h	v	w	u	h	v
还原的信息	v	i	s	a	i	n	P	a	r	t	e	s	t	r	e	s

凯撒大帝提供的这种加密解密算法，简单而实用，被称为凯撒密码。从现代的观点来看，凯撒密码仅仅是单表代换密码系统的一个特例。

早期的密码学能够体现出一种艺术之美，但安全性不高、科学性不强。

1948 年，香农(Claude Shannon)发表了一篇具有里程碑意义的论文——"通信的数学理论"(*A Mathematical Theory of Communication*)，宣告了信息论的诞生。紧接着他又发表了一篇名为"保密系统的通信理论"(*Communication Theory of Secrecy Systems*)的论文，该论文用信息论的观点全面地论述信息保密问题，将密码学的研究从古典的朴素研究引入到科学研究的轨道上，同时香农理论也为对称密码学的构造提供了直接的理论基础。从此，密码学成为一门真正的科学。

图 2 是 Shannon 在论文中提出的通用保密系统方案，方案中已明确引入了攻击者(Enemy Cryptanalyst)的角色。

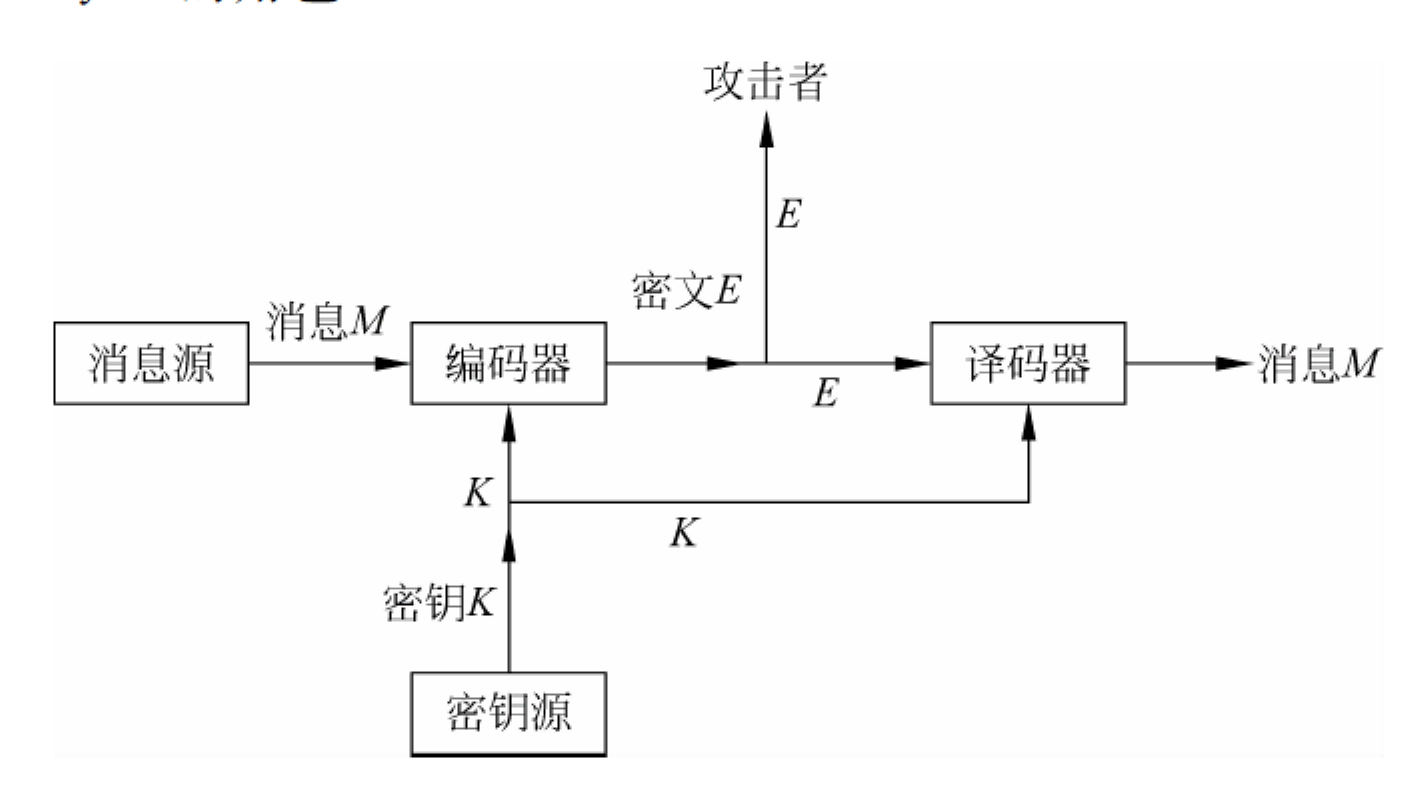


图 2　Shannon 的通用保密系统方案

密码学沿着对称密码体制的方向缓慢发展，直到公钥密码体制的突然出现。1976 年，Diffie 与 Hellman 合作发表了一篇经典论文"密码学的新方向"(*New Directions in Cryptography*)，提出一种可以使通信双方在不安全信道上进行的安全密钥协商协议(Diffie-Hellman 协议)，并在此基础上提出了公钥密码算法的思想和概念。这篇论文也标志着传统密码学开始向现代密码学的转变。

图 3 是 Diffie 与 Hellman 在论文中提出的公钥密码系统方案，使用两个不同的密钥分别用于加密和解密，从而不必再维护传递密钥的信道，这是密码学的一项创举。

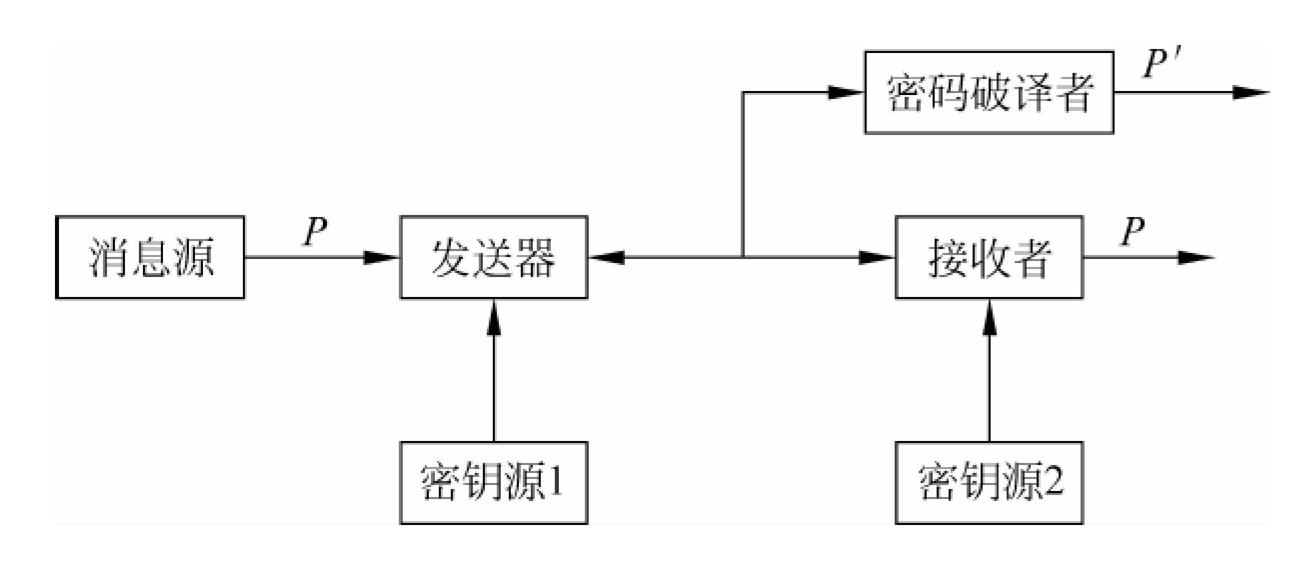


图 3　Diffie 与 Hellman 的公钥密码系统方案

1978 年，Rivest、Shamir 和 Adleman 基于大整数因式分解难题，提出第一种实用的公钥密码体制方案，称为 RSA 密码体制。此后，陆续有新的公钥密码体制被提出，如 1978 年提出的基于背包问题的 Merkle-Hellman 背包公钥密码体制、1985 年提出的基于离散对数难题的 ElGamal 公钥密码体制。

公钥密码体制的作用并不局限于加密解密，它的另外一个重要贡献是数字签名。1991年，第一个关于数字签名的标准(ISO/IEC 9796)应运而生。起初的标准基于RSA，但1994年后基于ElGamal的数字签名更受青睐。

随着计算机科学与技术的发展，密码学在信息安全领域也发挥出越来越重要的作用。无论是数据加密、防篡改还是身份验证，到处都能看到密码学的应用。

本书正是按照密码学发展的过程和思路，由浅入深地介绍密码学的基本理论。在此基础上，进一步介绍密码学在计算机安全中的应用。

本书的内容编排如下：

第1章　从古典密码系统开始介绍密码学的基本概念和历史上的一些经典算法。

第2章　重点讲解DES、AES等对称密码系统，以及ECB、CBC等加密解密模式。

第3章　讲解基于大数因式分解问题的RSA、Rabin公钥密码系统。

第4章　讲解基于离散对数问题的ElGamal、ECC公钥密码系统。

第5章　介绍包括Knapsack、McEliece、NTRU等在内的其他公钥密码系统。

第6章　重点介绍数字签名的原理。

第7章　介绍Hash函数以及基于Hash函数和对称密码算法的消息验证码技术。

第8章　介绍密钥的管理以及PKI技术。

第9章　以专题的形式讲解目前计算机安全的一些技术，包括口令安全、VPN等。

第10章　基于Java平台，介绍相关的密码学和安全编程框架和技术。

为了让读者掌握计算机安全和保密的相关知识和思想，在学习方法上给出以下建议：

(1) 尽可能深入地学习本书的数学基础课程：抽象代数。无论是对称密码体制的标准AES，还是众多的公钥密码体制，大多数的算法都建立在抽象代数基础之上。

(2) 要理解本书中的各种概念、结论和算法，实践必不可少。大多数情况下，本书提供了具体的例子。但即便如此，读者还应该自己更换参数，多练多算多想。

(3) 计算机专业的读者最好通过上机编程来实现书中的算法，以增强对算法的理解。由于有些算法过于复杂不易调试，读者可以参考书中一些实际算法的中间过程。

第 1 章　古典密码系统

可以将香农在 1948 年发表的论文“保密系统的通信理论”作为里程碑，之前的密码学统称为古典密码学。古典密码算法花样繁多，简单而有趣。

1.1　密码系统基本概念

在介绍古典密码算法之前，首先对各种古典密码算法进行一个简单的分类。

从加密手段分类，古典密码算法有换位(Transposition)密码和代换(Substitution)密码两种。

换位密码算法通过对明文的字母位置进行交换实现加密。

例 1.1　helloworld→hleolrowld。

观察字母的位置变化，可以看到明文的第 2 个字母与明文的第 3 个字母交换，明文的第 4 个字母与明文的第 5 个字母交换，明文的第 6 个字母与明文的第 8 个字母进行交换。

为了便于分析，本章中对于古典密码算法中的字母都进行统一的编码。

换位密码编码规则如表 1.1 所示。

表 1.1　换位密码编码规则

字母	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
数字	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

本书中忽略字母的大小写，如果不忽略，将有 52 个编码。

按照编码规则，可以对 helloworld 进行编码，即(7,4,11,11,14,22,14,17,11,3)。

假设明文长度为 10，那么所有长度为 10 的明文有 26^{10} 种，于是称这 26^{10} 种明文构成的集合形成明文空间。

经过换位后，得到的密文长度也为 10，所有密文的可能性也有 26^{10} 种，于是称这 26^{10} 种密文构成的集合形成密文空间。

也可以用矩阵运算来表示换位密码的规则：

设明文 x=(7,4,11,11,14,22,14,17,11,3)，则密文

$$y = x \cdot P_\pi$$

可见，在这个算法中，P_π 是加密的关键要素，称为密钥。所有可能的密钥也构成一个集合，这个集合形成密钥空间。

接收者得到密文后，需要解密。在换位密码算法中，解密是加密的换位逆过程。具体做法是密文的第 3 个字母与密文的第 2 个字母交换；密文的第 5 个字母与密文的第 4 个字母

交换；密文的第 8 个字母与密文的第 6 个字母交换。

当然解密过程也可以通过矩阵运算来表示，即：

$$x = y \cdot P_{\pi}^{-1}$$

无论是哪种密码系统，一般都包含上面提到的要素。

下面给出密码系统的定义：

一个加密与解密的完整体系称为密码系统或密码体制(Crypto System)。一个密码体制由明文空间 P、密文空间 C、密钥空间 K、加密算法集 E 和解密算法集 D 组成的五元组 (P,C,K,E,D) 构成。

(1) 明文空间 P：作为加密输入的原始信息 m 称为明文。所有可能明文的集合称为明文空间，通常用 P 表示。

(2) 密文空间 C：明文经加密变换后的数据 c 称为密文。所有可能密文的集合称为密文空间，通常用 C 表示。

(3) 密钥空间 K：密钥 k 是参与密码变换的参数。一切可能密钥构成的集合称为密钥空间(Keyspace)，通常用 K 表示。

(4) 加密算法集 E：任意给定密钥 $k \in K$，在加密算法集 E 中存在唯一的加密算法 $e_k \in E: P \to C$，将明文 $p \in P$ 变换为密文 $c \in C$。对明文 p 进行变换的过程称为加密(Encryption)。

(5) 解密算法集 D：任意给定密钥 $k \in K$，在加密算法集 D 中存在唯一的解密算法 $d_k \in D: C \to P$，将密文 $c \in C$ 变换为明文 $p \in P$。对密文 c 进行变换的过程称为解密(Decryption)。

有了数学概念后，下面重新来定义换位密码系统。

换位密码系统：明文空间和密文空间都是 Z_n^m，明文用$(x_1, x_2, \cdots, x_m)$表示，密文用$(y_1, y_2, \cdots, y_m)$表示，设密钥为 k，则：

加密算法：

$$(y_1, y_2, \cdots, y_m) = (x_{k(1)}, x_{k(2)}, \cdots, x_{k(m)})$$

解密算法：

$$(x_1, x_2, \cdots, x_m) = (y_{k^{-1}(1)}, y_{k^{-1}(2)}, \cdots, y_{k^{-1}(m)})$$

其中，k^{-1}是 k 的逆。

Z_n 表示集合$\{0,1,\cdots,n-1\}$。

例 1.2 在换位密码算法中，明文是“olympic”，求其密文。

写成编码的形式为：

$$(14,11,24,12,15,8,2)$$

设密钥为 k，k 可以用一个换位矩阵来表示：

$$P_{\pi} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

加密算法：

$$(y_1, y_2, y_3, y_4, y_5, y_6, y_7) = xP_\pi = (x_2, x_4, x_7, x_3, x_6, x_1, x_5)$$

生成密文的编码：

$$(11,12,2,24,8,14,15)$$

即“lmcyiop”。

解密算法：

$$(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = yP_\pi^{-1} = (y_6, y_1, y_4, y_2, y_7, y_5, y_3)$$

还原出明文“olympic”。

1.2　移位密码系统

与换位密码系统不同，代换密码系统中的密文与明文的关系不是位置上的变化，而是编码上的变化。

根据密码系统加解密时使用替换表多少不同，代换密码系统可以分为单表代换密码系统和多表代换密码系统。

先来看移位密码系统，这是一种典型的单表代换密码系统。

所谓移位密码系统，是指明文的所有字母后移一定的数量形成密文的字母。

表 1.2 是从明文字母映射到密文字母的一个实例。

表 1.2　明文字母到密文字母的映射

明文	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
密文	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f

有了映射关系后，就可以进行加密和解密。

例 1.3　根据表 1.2，对明文“helloworld”进行加密。

按照表 1.2 的映射关系，存在以下的加密变换：

$$h \to n, e \to k, l \to r, l \to r, o \to u, w \to c, o \to u, r \to x, l \to r, d \to j$$

于是可以得到密文：

“nkrrucuxrj”

接收方收到密文后，也可以通过表 1.2 进行还原：

$$n \to h, k \to e, r \to l, r \to l, u \to o, c \to w, u \to o, x \to r, r \to l, j \to d$$

得到明文：

“helloworld”

移位密码系统的思想实质上是在字母编码上加上某一个数，因此也称为加法密码系统。下面对仿射移位密码系统进行完整的描述。

移位密码系统：在移位密码系统(Shift Cipher)中，明文空间和密文空间均为 Z_n。

密钥空间：　$K = \{k,\ k \in Z_n\}$

加密算法：　$E_k(i) = (i + k) \bmod n$

解密算法： $D_k(j)=(j-k) \bmod n$

引言提到的凯撒密码是移位密码系统的一个特例，$k=3$ 时的移位密码系统就成了凯撒密码系统。

既然移位密码系统又称加法密码系统，是否可以考虑把加法变成乘法，形成一种新的密码系统。答案是肯定的，这种密码系统称为乘法密码系统。

同样是明文“helloworld”，对应的编码是 $x[10]=(7,4,11,11,14,22,14,17,11,3)$，如果要用乘法密码算法加密，假设 $k=3$，那么如何来求 $y[10]$呢？

把对应的每个数乘以3，得到(21,12,33,33,42,66,42,51,33,9)，为了让所有元素都能表示成26个字母，必须在乘法完成后取模26，不难得到：

$$y[10]=(21,12,7,7,16,14,16,25,7,9)$$

即“vmhhqoqzhj”。

但是，如果要对 $y[10]=(21,12,7,7,16,14,16,25,7,9)$解密，就会遇到一点麻烦，如：

$$7/3 \bmod 26 = ?$$

这是乘法模逆求解问题，这类问题的解决方案是欧几里德扩展算法（Euclidean Algorithm）。

1.3 欧几里德扩展算法

在介绍欧几里德扩展算法之前，先回顾一些基本的数学概念。

最大公约数(Greatest Common Divisor，gcd)：如果有一个自然数 a 能被自然数 b 整除，则称 a 为 b 的倍数，b 为 a 的约数。几个自然数共有的约数，称为这几个自然数的公约数。公约数中最大的一个公约数，称为这几个自然数的最大公约数。

例 1.4 在2、4、6中，2就是2、4、6的最大公约数。

等价关系(Equivalence Relation)：设～是集合 G 上的一个二元关系，满足以下条件：

自反性：对任何 $a\in G$，有 $a\sim a$；

对称性：对任何 a、$b\in G$，有 $a\sim b\Rightarrow b\sim a$；

传递性：对任何 a、b、$c\in G$，有 $a\sim b$ 且 $b\sim c\Rightarrow a\sim c$；

则称～为 G 中的一个等价关系。

同余关系(Congruence Relation)：设 m 为非零整数，a、b 为整数，如果 $m|a-b$，那么称 a 与 b 对模 m 是同余关系，记作：

$$a = b \bmod m$$

不难验证，同余关系是一种等价关系，它满足等价关系的自反性、对称性和传递性。

自反性：对于任何整数 a，a 与 a 对模 m 同余。

对称性：对于任何整数 a、b，如果 a 与 b 对模 m 同余，那么显然有 b 与 a 对模 m 同余。

传递性：对于任何整数 a、b、c，如果 a 与 b 对模 m 同余，且 b 与 c 对模 m 同余，那么 a 与 c 对模 m 同余。

等价类(Equivalence Class)：子集$\bar{a}=\{x \mid x\in G, x\sim a\}$称为 a 所在的等价类。

换句话说，等价类就是与 a 等价的元素的集合。

定义在整数集 Z 上的同余关系，可以将整数集 Z 划分为“同余类”。

同余类(Congruence Class)：称集合$\{b: b\equiv a(\bmod\ m)\}$为模 m 的同余类，又称同余子集，记为$[a]_m$，如果 $a_0\in[a]_m$，且 $0\leqslant a_0<m$，则称 a_0 为$[a]_m$ 的主余数，记为$\langle a\rangle_m$。

主余数$\langle a\rangle_m$ 属于集合 Z_m，$Z_m=\{1,2,\cdots,m-1\}$。

在 Z_m 中可以定义加法运算，例如($m=26$ 时)：

$$2+3=5 \bmod 26$$

$$22+7=3 \bmod 26$$

也可以定义乘法运算，例如：

$$2\times 3=6 \bmod 26$$

$$22\times 7=24 \bmod 26$$

加法的逆运算是减法运算，例如：

$$5-3=2 \bmod 26$$

$$3-22=7 \bmod 26$$

在计算 $3-22 \bmod 26$ 时，可以先得到-19，然后找主余数 7 得到结果。

乘法的逆运算是除法运算，例如：

$$6/3=2 \bmod 26$$

当要计算 $24/7 \bmod 26=?$ 时，却出现了困难。

定理 1.1(**欧几里德算法**)　两个正整数 $r_0>r_1$，欧几里德算法包括下列分式组：

$$\begin{aligned}
r_0 &= q_1 r_1 + r_2 \quad 0<r_2<r_1\\
r_1 &= q_2 r_2 + r_3 \quad 0<r_3<r_2\\
&\vdots\\
r_{m-2} &= q_{m-1} r_{m-1} + r_m \quad 0<r_m<r_{m-1}\\
r_{m-1} &= q_m r_m
\end{aligned}$$

通过欧几里德算法可以得出 r_0 和 r_1 的最大公约数 r_m：

$$\gcd(r_0,r_1)=\gcd(r_1,r_2)=\cdots=\gcd(r_{m-1},r_m)=r_m$$

例 1.5　求解 gcd(2001,99)。

$$\begin{aligned}
&2001=20\times 99+21\\
&99=4\times 21+15\\
&21=1\times 15+6\\
&15=2\times 6+3\\
&6=2\times 3\\
&r_m=3
\end{aligned}$$

因此，gcd(2001,99)=gcd(99,21)=…=gcd(6,3)=3。

定理 1.2(**欧几里德扩展算法**)　根据欧几里德算法的参数$(q_1,q_2,\cdots,q_m)$，构造序列

$$t_0,t_1,\cdots,t_m$$

$$t_0 = 0$$
$$t_1 = 1$$
$$\cdots$$
$$t_j = t_{j-2} - q_{j-1}t_{j-1} \bmod r_0$$

对于 $0 \leqslant j < m, r_j \equiv t_j r_1 \bmod r_0$ 恒成立。

证明：容易验证当 $j=0$ 和 $j=1$，等式 $r_j \equiv t_j r_1 \bmod r_0$ 成立。

假设当 $j=i-2$ 和 $j=i-1$ 时等式成立，即：

$$r_{i-2} \equiv t_{i-2}r_1 \bmod r_0$$
$$r_{i-1} \equiv t_{i-1}r_1 \bmod r_0$$

那么：

$$\begin{aligned} r_i &= r_{i-2} - q_{i-1}r_{i-1} \\ &= t_{i-2}r_1 - q_{i-1}t_{i-1}r_1 \\ &= (t_{i-2} - q_{i-1}t_{i-1})r_1 \\ &\equiv t_i r_1 \pmod{r_0} \end{aligned}$$

根据数学归纳法，$r_j \equiv t_j r_1 \bmod r_0$ 恒成立，命题得证。

定理 1.3 根据前面的定义，并设 $\gcd(r_0, r_1)=1$，则 $t_m = r_1^{-1} \bmod r_0$。

证明：因为：

$$r_m = \gcd(r_0, r_1) = 1$$

令 $j=m$，有：

$$t_m r_1 \equiv r_m \equiv 1 \bmod r_0$$

因此，$t_m = r_1^{-1} \bmod r_0$ 成立。命题得证。

利用欧几里德扩展算法可计算元素的逆，算法描述如下：

```
a = r0
b = r1
c = 0
d = 1
q = [a/b]
r = a - q×b
while r > 0 do
    temp = c - q×d
    if temp > 0 then temp = temp mod a
    else temp = n - temp mod a
    c = d
    d = temp
    a = b
    b = r
    q = [a/b]
    r = a - q×b
if b ! = 1 then r1 的模 r0 逆不存在
else d 是 r1 的模 r0 逆
```

例 1.6　求解 7^{-1} mod 26。

解：

$$26 = 3 \times 7 + 5,\ t_0 = 0$$
$$7 = 1 \times 5 + 2,\ t_1 = 1$$
$$5 = 2 \times 2 + 1,\ t_2 = t_0 - q_1 t_1 = 0 - 3 \times 1 = 23 (\text{mod } 26)$$
$$2 = 2 \times 1,\ t_3 = t_1 - q_2 t_2 = 1 - 1 \times 23 = 4 (\text{mod } 26)$$
$$7^{-1} = t_4 = t_2 - q_3 t_3 = 23 - 2 \times 4 = 15 (\text{mod } 26)$$

容易验证：

$$7 \times 15 = 105 \equiv 1 (\text{mod } 26)$$

例 1.7　求解 24/7 mod 26。

解：

$$24/7 = 24 \times 7^{-1}$$
$$= 360$$
$$= 22 \text{ mod } 26$$

有了欧几里德扩展算法后，乘法密码系统的解密问题便迎刃而解。

乘法密码系统： 在乘法密码系统（Multiplication Cipher）中，明文空间和密文空间均为 Z_n。

密钥空间：　$K = \{k \mid \gcd(k, n) = 1,\ k \in Z_n\}$

加密算法：　$E_k(i) = (i \cdot k) \text{ mod } n$

解密算法：　$D_k(j) = (j/k) \text{ mod } n$

这里需要注意的是乘法密码系统中的密钥需满足一个条件：$\gcd(k,n)=1$，即 k 与 n 互素。这个条件也是乘法模逆存在的充分必要条件。

将加法密码系统和乘法密码系统结合起来，能够形成仿射密码系统。

仿射密码系统： 在仿射密码系统（Affine Cipher）中，明文空间和密文空间均为 Z_n。

密钥空间：　$K = \{k = (k_0, k_1) \mid \gcd(k_1, n) = 1,\ k_0, k_1 \in Z_n\}$

加密算法：　$E_k(i) = (ik_1 + k_0) \text{ mod } n$

解密算法：　$D_k(j) = (j - k_0)/k_1 \text{ mod } n$

与前面的密码系统相比，仿射密码系统的密钥不再是一个数，而是一个二元组 (k_0, k_1)。

例 1.8　在仿射密码系统中，明文空间和密文空间均为 Z_{26}，$k_0=3$，$k_1=5$。

英文明文：

"unconditional security"

相应的数字明文：

"20,13,2,14,13,3,8,19,8,14,13,0,11,18,4,2,20,17,8,19,24"

实施加密算法：

$$E_k(i) = (5i + 3) \text{ mod } 26$$

得到数字密文：

"25,16,13,21,16,18,17,20,17,21,16,3,6,15,23,13,25,10,17,20,19"

相应的英文密文：

"zqnvqsrurvqdg pxnzkrut"

实施解密算法：

$$D_k(j) = (j-3)/5 \bmod 26$$

还原出数字明文：

"20,13,2,14,13,3,8,19,8,14,13,0,11,18,4,2,20,17,8,19,24"

还原出英文明文：

"unconditional security"

特别地，当 $k_1=1$ 时，仿射退化成简单的加和减，相应的密码系统是移位密码系统；当 $k_0=0$ 时，仿射退化成乘法，相应的密码系统是乘法密码系统。

无论是移位密码系统、乘法密码系统还是仿射密码系统，都是一种字符的替换。以 $k_0=3, k_1=5$ 的仿射密码系统为例，加密算法对应的密码表，如表 1.3 所示。

表 1.3　仿射密码系统的密码表

明文	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
密文	d	i	n	s	x	c	h	m	r	w	b	g	l	q	v	a	f	k	p	u	z	e	j	o	t	y

因此，这些密码算法都是单表代换密码系统。

单表代换密码系统还有更一般的形式，如表 1.4 所示。

表 1.4　单表代换密码系统的密码表

明文	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
密文	o	q	f	s	j	n	t	a	g	v	i	c	e	x	p	d	y	m	z	r	b	k	u	h	w	l

一般形式下的单表代换密码系统的加密和解密算法通常需要通过查表来实现，而没有四则运算那么方便。与一般形式下的单表代换密码系统相比，密码表的基数越大，仿射密码系统的运算优势越明显。当然，一般形式下的单表代换密码系统显然更安全一些。

1.4　单表代换密码分析

本节的主要内容是对于单表代换密码系统的安全性进行分析。

考虑英文字母的移位密码系统，由于其密钥量很小，即使通过穷举密钥搜索也不需多少开销，因此密码系统安全的一个前提条件是密钥空间足够大。

除了密钥空间要考虑之外，单表代换密码系统的另一个缺点是没有很好的隐藏字母的频率规律，因此一般运用频率统计方法可以得到单表代换密码系统的密钥。

几乎每一种语言都有其基础元素的频率规律。以英语为例,26 个字母中 e 和 t 出现的频率较高,x 和 z 出现的频率则较低,具体的统计数据如表 1.5 所示。

表 1.5 英文字母概率表

字母	概率	字母	概率
a	0.082	n	0.067
b	0.015	o	0.075
c	0.028	p	0.019
d	0.043	q	0.001
e	0.127	r	0.060
f	0.022	s	0.063
g	0.020	t	0.091
h	0.061	u	0.028
i	0.070	v	0.010
j	0.002	w	0.023
k	0.008	x	0.001
l	0.040	y	0.020
m	0.024	z	0.001

根据表 1.5 可知,英语中英文字母出现的频率有很大的差异,为了进一步的密码分析,通常将 26 个字母按频率高低分为以下几组:

第一组,e 的出现频率最大。

第二组,t、a、o、i、n、s、h、r 的频率略低于 e,称为次高频组。

第三组,d、l 为中频组。

第四组,c、u、m、w、f、g、y、p、b 为低频组。

第五组,v、k、j、x、q、z 为极低频组。

另外,还有一些频繁出现的双字母组和三字母组的频率统计也有助于密码分析。

按频率从高到低排序,以下为 30 个最常出现的双字母组:

th,he,in,er,an,re,de,on,es,st,en,at,to,nt,ha,nd,ou,ea,ng,as,or,ti,is,et,it,ar,te,se,hi,of

以及 12 个最常出现的三字母组:

the,ing,and,her,ere,ent,tha,nth,was,eth,for,dth

这种频率统计的特性,通过单表代换的加密后并没有改变,因此通过相应频率规律便可还原出原文。

例 1.9 对于给定的如下密文:

yifqfmzrwqfyvecfmdzpcvmrzwnmdzvejbtxcddumj
ndifefmdzcdmqzkceyfcjmyrncwjcszrexchzunmxz
nzucdrjxyysmrtmeyifzwdyvzvyfzumrzcrwnzdzjj
xzwgchsmrnmdhncmfqchzjmxjzwiejyucfwdjnzdir

先统计出各个字母的出现次数,具体如表 1.6 所示。

表 1.6　字母出现次数统计表

字母	次数	字母	次数
a	0	n	9
b	1	o	0
c	15	p	1
d	13	q	4
e	7	r	10
f	11	s	3
g	1	t	2
h	4	u	5
i	5	v	5
j	11	w	8
k	1	x	6
l	0	y	10
m	16	z	20

通过表 1.6 可以看到，在密文中字母 z 出现频率明显高于其他字母，因此可以考虑 z 对应的明文字母是 e。其他的字母在密文中出现频率较高的字母是 c、d、f、j、m、r、y，它们出现次数都在 10 次以上，因此可以考虑在次高频组中对应的明文字母，不过具体的对应关系尚不能轻易确定。

可以进一步分析双字母组合出现的频率。

假设 z 对应的明文字母是 e，那么先分析-z 或 z-型的双字母组。其中，出现较多的是 dz 和 zw 分别为 4 次，nz 和 zu 出现 3 次，rz、hz、xz、fz、zr、zv、zc、zd 和 zj 各出现两次。由现有数据可以看出，zw 出现 4 次而 wz 一次都未出现，同时 w 出现的次数要低于已假设的次高频组的字母出现次数，因此可以考虑 w 对应的明文字母为 d。

又因为 dz 出现 4 次而 zd 出现两次，且 d 本身出现频率也较高，因此可以考虑 d 对应的明文字母可是 r、s、t 中的某一个。

通过前面的分析，z 可能对应字母 e，w 可能对应字母 d，zrw 和 rzw 在密文的开始部分有出现，rw 在后文中也出现了。而 r 已经考虑为次高频组中的字母，同时 nd 是一个明文常见的双字母组，因此可以考虑 r 对应明文字母 n。

将以上的分析，转化为对应明文，具体形式如下：

```
- - - - - - e n d - - - - - - - - - e - - - - n e d - - - e - - - - - - - - - - - -
y i f q f m z r w q f y v e c f m d z p c v m r z w n m d z v e j b t x c d d u m j
- - - - - - - - e - - - - e - - - - - - - - - n - - d - - - e n - - - - e - - - - e
n d i f e f m d z c d m q z k c e y f c j m y r n c w j c s z r e x c h z u n m x z
- e - - - n - - - - - - n - - - - - - e d - - - e - - - e - - n e - n d - e - e - -
n z u c d r j x y y s m r t m e y i f z w d y v z v y f z u m r z c r w n z d z j j
- e d - - - - - n - - - - - - - - - - - e - - - - e d - - - - - - - d - - - e - - n
x z w g c h s m r n m d h n c m f q c h z j m x j z w i e j y u c f w d j n z d i r
```

接下来，可以考虑密文中的 n 对应明文字母 h，这是因为在密文中 nz 出现次数较多而 zn 出现次数较少，这符合 he 与 eh 的字母频率统计规律。在此假设基础上，密文中的

rzcrwnz 可能对应明文 ne-ndhe，而由此明文可以考虑 c 对应明文 a，则明文可能形式为：

```
- - - - - - e n d - - - - - a - - - e - a - - n e d h - - e - - - - - - a - - - - -
y i f q f m z r w q f y v e c f m d z p c v m r z w n m d z v e j b t x c d d u m j
h - - - - - - - e a - - - e - a - - - a - - - n h a d - a - e n - - a - e - h - - e
n d i f e f m d z c d m q z k c e y f c j m y r n c w j c s z r e x c h z u n m x z
h e - a - n - - - - - - n - - - - - - e d - - - e - - - e - - n e a n d h e - e - -
n z u c d r j x y y s m r t m e y i f z w d y v z v y f z u m r z c r w n z d z j j
- e d - a - - - n h - - - h a - - - a - e - - - - e d - - - - - a - d - - h e - - n
x z w g c h s m r n m d h n c m f q c h z j m x j z w i e j y u c f w d j n z d i r
```

现在再考虑另一个出现次数较高的密文字母 m，根据前文的分析可得，密文段 rnm 解密成 nh-，这表明 h-是一个词的开头，因此 m 很可能是一个元音。

而根据前文的分析已假定了 a 和 e，这里可以考虑 m 是次高频组中的元音字母 i 或 o。同时，在英语中双字母 ai 比 ao 的出现频率高，因此首先可以考虑 m 是密文字母 i，这样明文可能的形式为：

```
- - - - - i e n d - - - - - a - i - e - a --i n e d h - - e - - - - - - a - - - i -
y i f q f m z r w q f y v e c f m d z p c v m r z w n m d z v e j b t x c d d u m j
h - - - - - i --e a - i - e - a - - - a - i --n h a d - a - e n - - a - e - h i - e
n d i f e f m d z c d m q z k c e y f c j m y r n c w j c s z r e x c h z u n m x z
h e - a - n - - - - - i n - i - - - - e d - - - e - - - e --i n e a n d h e - e - -
n z u c d r j x y y s m r t m e y i f z w d y v z v y f z u m r z c r w n z d z j j
- e d - a - - i n h i - - h a i - - a - e - i - - e d - - - - - a - d - - h e - - n
x z w g c h s m r n m d h n c m f q c h z j m x j z w i e j y u c f w d j n z d i r
```

继续分析密文，确定明文字母 o 对应的可能密文字母。因为 o 是一个经常出现的明文字母，所以可以考虑密文中的字母 d、f、j、y。

经过对密文的观察，y 的可能性较大，否则会出现长串的元音字母，因为密文中有字段 cfm 或 cjm，若为对应明文字母 o，则出现了 aoi，这不符合英语中的语法习惯。因此，可以考虑字母 y 对应明文字母 o。

剩余的密文字母中出现频率最高的 3 个字母是 d、f、j，可以考虑它们以某种次序解密成 r、s、t，三字母 nmd 两次出现，很可能密文 d 对应明文字母 s，因为可能是明文三字母组 his。

对于密文字母段 hncmf，由于已假设 n 为 h、c 为 a、m 为 i，所以可以猜测明文字母段是 chair，这样可以假设密文 f 对应明文字母 r、密文 h 对应明文字母 c，而最后一个密文中出现频率较高的 j 假设为明文字母 t。这样对应的明文形式为：

```
o - r - r i e n d - r o - - a r i s e - a - i n e d-h i s e - - - t - - a - - - i t
y i f q f m z r w q f y v e c f m d z p c v m r z w n m d z v e j b t x c d d u m j
h s - r - r i-s e a s i --e--a-o r a t i - o n h a d t a - e n - - a c e --h i - e
n d i f e f m d z c d m q z k c e y f c j m y r n c w j c s z r e x c h z u n m x z
h e - a s n t - o o --i n - i --o- r e d s o - e --o r e - i n e a n d h e s e t t
n z u c d r j x y y s m r t m e y i f z w d y v z v y f z u m r z c r w n z d z j j
- e d --a c --i n h i-s c h a i-r--a c e t i - t e d - - t o - a r d s t h e s - n
x z w g c h s m r n m d h n c m f q c h z j m x j z w i e j y u c f w d j n z d i r
```

根据以上的分析结果,已经基本确定出了大部分明文,接下来就是将剩余的密文字母根据语义与词法填补完整,即可得到明文。具体的解密明文如下:

our friend from paris examined his empty glass with surprise, as if evaporation had taken place while he wasn't looking. i poured some more wine and he settled back in his chair, face tilted up to the sun.

显然,字母频率分析法是单表代换密码系统的破解利器。

在对单表代换密码系统进行分析之后,下面对密码系统的分析(攻击)进行一次简单的分类。根据密码分析者对密码系统的掌握程度,可将密码系统分析类型分为以下几种:

1) 唯密文攻击(Ciphertext-Only Attack)

对于这种形式的密码分析,分析者已知两个信息:加密算法和待破译的密文。密码分析者的任务是恢复尽可能多的明文,或是直接推算出加密消息的密钥,以便将来采用相同的密钥解出其他被加密的消息。

2) 已知明文攻击(Known-Plaintext Attcak)

对于这种形式的密码分析,分析者掌握的信息包括加密算法和经密钥加密形成的一个或多个明文-密文对,即知道一定数量的密文和对应的明文。其任务是利用加密的信息推出用来加密的密钥或导出一个算法,该算法可以对用同一密钥加密的任何新的消息进行解密。

3) 选择明文攻击(Chosen-Plaintext Attack)

分析者除了知道加密算法外,还可以选定明文消息,并可以知道对应的加密得到的密文,即知道选择的明文和对应的密文。这种情形特别适用于后面章节将介绍的公钥密码系统,因为分析者可以利用公钥加密任意选定的明文。

一个密码体制是否安全,通常是指在这 3 种攻击下的安全性,因为攻击者很容易具备进行 3 种攻击的条件。

相对而言,唯密文攻击是最困难的,因为分析者可供利用的信息最少。单表代换密码系统的字母频率分析法正是一种唯密文攻击手段。

1.5 Vigenère 密码系统

对单表代换密码系统的改进是多表代换密码系统,多表代换密码系统的代表是 Vigenère 密码系统。Vigenère 密码系统由法国密码学家 Blaise de Vigenère 于 1858 年提出的一种算法。

Vigenère 密码系统:在 Vigenère 密码系统中,明文空间和密文空间均为 Z_n。

密钥: $k=(k_0,k_1,\cdots,k_d)$, $k_i\in Z_n$, $i=0,1,2,\cdots,d$

加密算法:

$$m_{i+td}\rightarrow c_{i+td}$$

$$c_{i+td}=(m_{i+td}+k_i)(\bmod n)$$

解密算法:

$$c_{i+td}\rightarrow m_{i+td}$$

$$m_{i+td}=(c_{i+td}-k_i)(\bmod n)$$

例 1.10 在 Vigenère 密码系统中,明文和密文都是英文字母串,$n=26$,密钥 $k=\{19,7,8,13,10\}$。

明文的英文形式是：

"the algebraic form of an elliptic curve"

相应的数字形式是：

"19,7,4,0,11,　6,4,1,17,0,　8,2,5,14,17,　12,14,5,0,13,
4,11,11,8,15,　19,8,2,2,20　17,21,4"(忽略空格)

实施加密算法：

19(+19)→12	7(+7)→14	4(+8)→12	0(+13)→13	11(+10)→21
6(+19)→25	4(+7)→11	1(+8)→9	17(+13)→4	0(+10)→10
8(+19)→1	2(+7)→9	5(+8)→13	14(+13)→1	17(+10)→1
12(+19)→5	14(+7)→21	5(+8)→13	0(+13)→13	13(+10)→23
4(+19)→23	11(+7)→18	11(+8)→19	8(+13)→21	15(+10)→25
19(+19)→12	8(+7)→15	2(+8)→10	2(+13)→15	20(+10)→4
17(+19)→10	21(+7)→2	4(+8)→12		

得到数字密文：

"12,14,12,13,21,　25,11,9,4,10,　1,9,13,1,1,　5,21,13,13,23,
23,18,19,21,25,　12,15,10,15,4,10,2,12"

相应的英文密文：

"mom nvzljekbj nbbf vn nx xstvzmpk pekcm"

实施解密算法：

12(−19)→19	14(−7)→7	12 (−8)→4	13(−13)→0	21(−10)→11
25(−19)→6	11(−7)→4	9(−8)→1	4(−13)→17	10(−10)→0
1(−19)→89	(−7)→2	13(−8)→5	1(−13)→14	1(−10)→17
5(−19)→12	21(−7)→14	13(−8)→5	13(−13)→0	23(−10)→13
23(−19)→4	18(−7)→11	19(−8)→11	21(−13)→8	25(−10)→15
12(−19)→19	15(−7)→8	10(−8)→2	15(−13)→2	4(−10)→20
10(−19)→17	2(−7)→21	12(−8)→4		

还原出数字明文：

"19,7,4,0,11,　6,4,1,17,0,　8,2,5,14,17,　12,14,5,0,13,
4,11,11,8,15,　19,8,2,2,20,　17,21,4"

相应的英文明文是：

"the algebraic form of an elliptic curve"

在 Vigenère 密码系统中，具体密钥 $k=(k_1, k_2, k_3, \cdots, k_n)$的选取可以通过一个英文单词来设定，即将$(k_1, k_2, k_3, \cdots, k_n)$与这个单词的每一个字母一一对应。

由加密过程可见，对于同一个字母的加密，可能会对应不同的密文，也就是说字母对照表不是一个，而是多个，因此 Vigenère 密码系统是一种多表代换密码系统。

多表代换密码系统在一定程度上隐藏了字母的频率规律，对于单表代换密码的频率分析方法在多表代换密码系统分析中便不能直接适用。要分析 Vigenère 密码，首先需确定密钥的长度 n，较常用的方法有 Kasiski 测试法和重合指数法两种。

Kasiski 测试法是由普鲁士军官 Kasiski 在 1863 年提出的一种重码分析法。基本原理：

搜索长度至少为 3 的相同的密文段，记录这些相同密文段到起始点之间的距离，从而得到几个距离的数值，可以猜测 n 为这些距离值的最大公因子的因子。

Kasiski 测试过程，只是给出了 n 值的初步猜测值，为得到更准确地判断，需要进一步结合重合指数法。

重合指数：设 $x=x_1x_2\cdots x_n$ 是 n 个字母的一个串，x 的重合指数定义为 x 中的两个随机元素相同的概率，记为 $I_c(x)$。

假设 f_0、f_1、…、f_{25} 分别表示 x 中的各个字母(A，B，C，…，Z)出现的频率。一共有 C_n^2 种方法选择 x 中的任意两个元素。对每一个 i，$0\leqslant i\leqslant 25$，x 中的两个元素都被选择为第 i 个字母的方法有 $C_{f_i}^2$ 种，于是可以得到：

$$I_c(x)=\frac{\sum_{i=0}^{25}C_{f_i}^2}{C_n^2}=\frac{\sum_{i=0}^{25}f_i(f_i-1)}{n(n-1)}$$

假设字母出现的期望概率为 p_0、p_1、p_2、…、p_{25}，两个随机元素都是 A 的概率为 p_0^2，两个随机元素都为 B 的概率为 p_1^2，依次类推。则有：

$$I_c(x)\approx\sum_{i=0}^{25}p_i^2=0.065$$

假定密文串 $c=c_1c_2c_3\cdots c_n$ 是由 Vigenère 密码加密得到的，把 c 分成 m 个长为 n/m 的子串，记为 y_1、y_2、…、y_m。如果 m 确实是密钥的长度，那么每一个 $I_c(y_i)$的值应该约等于 0.065。而如果 m 不是密钥的长度，那么子串应看起来更为随机，因为是通过不同的密钥移位加密方式得到的。而对于一个完全随机的字母串，其重合指数为：

$$I_c(x)\approx 26(1/26)^2=1/26\approx 0.038$$

对比两个重合指数的值 0.065、0.038，可以看出两者的差别相当大，因此可以用这个方法确定密钥的长度。

在确定密钥长度后，假定长度为 m，则可以把密文 c 分成 m 个长为 n/m 的子串，而对于每个子串实际上就相当于单表代换密码，可以通过对单表代换密码的分析方法来进行破解即可。不过，仍有一个更简单而有效的方法。

令 $1\leqslant i\leqslant m$，f_0、f_1、…、f_{25} 分别表示串 c_i 中的各个字母(A，B，C，…，Z)出现的频率。再令 $n'=n/m$ 表示串 c_i 的长度，则 26 个字母在 c_i 中出现的概率为$\frac{f_0}{n'}$、$\frac{f_1}{n'}$、…、$\frac{f_{25}}{n'}$。

考虑到子串 c_i 是由对应的待加密的明文子集中的字母移动 c_i 个位置所得的，因此移位后的概率分布为$\frac{f_{k_i}}{n'}$、$\frac{f_{k_i+1}}{n'}$、…、$\frac{f_{k_i+25}}{n'}$。

应该近似于表 1.5 中的统计概率分布 p_0、p_1、p_2、…、p_{25}。

假定 $0\leqslant g\leqslant 25$，定义数值：

$$M_g=\sum_{i=0}^{25}\frac{p_if_{i+g}}{n'}$$

如果 $g=k_i$，类似于前文的重合指数的分析，应该有：

$$M_g\approx\sum_{i=0}^{25}p_i^2=0.065$$

如果 $g \neq k_i$，则 M_g 一般应该小于 0.065。对于任意的 $i(1 \leqslant i \leqslant m)$，使用这种方法可以具体确定每一个 k_i 的值。

例 1.11　设某一段明文应用 Vigenère 密码加密后的密文如下：

chreevoahmaeratbiaxxwtnxbeeophbsbqmqeqerbwrvxuoakxaosxxweahbwgjmmqmnkgrf
vgxwtrzxwiaklxfpskautemndcmgtsxmxbtuiadngmgpsrelxnjelxvrvprtulhdnqwtwdtygbphxtfa
ljhasvbfxngllchrzbwelekmsjiknbhwrjgnmgjsglxfeyphagnrbieqjtamrvlcrremndglxrrimgnsnrw
chrqhaeyevtaqebbipeewevkakoewadremxmtbjjchrtkdnvrzchrclqohpwqaiiwxnrmgwoiifkee

首先，使用 Kasiski 测试法。在密文中密文串“chr”共出现在 5 个位置，开始位置分别为 1、166、236、276 和 286，其距离分别为 165、235、275 和 285。这 3 个整数的最大公约数为 5，故可以考虑密钥的长度很可能是 5。

然后，通过使用重合指数法再进一步确认这一分析的正确性。当 $m=1$ 时，重合指数为 0.045；当 $m=2$ 时，两个重合指数分别为 0.046 和 0.041；当 $m=3$ 时，3 个重合指数分别为 0.043、0.050 和 0.047；当 $m=4$ 时，4 个重合指数分别为 0.042、0.039、0.046 和 0.040；当 $m=5$ 时，可获得的值分别为 0.063、0.068、0.069、0.061 和 0.072。这些值为密钥的长度为 5 提供了强有力的证据。

在确定了密钥长度为 5 后，可以分析对任意 $1 \leqslant i \leqslant 5$，计算其对应的 M_g 的值。具体结果如表 1.7 所示。

表 1.7　M_g 的值

i	$M_g(C_i)$的值								
	0.035	0.031	0.036	0.037	0.035	0.039	0.028	0.028	0.048
1	0.061	0.039	0.032	0.040	0.038	0.038	0.044	0.036	0.030
	0.042	0.043	0.036	0.033	0.049	0.043	0.041	0.036	
	0.069	0.044	0.032	0.035	0.044	0.034	0.036	0.033	0.030
2	0.031	0.042	0.045	0.040	0.045	0.046	0.042	0.037	0.032
	0.034	0.037	0.032	0.034	0.043	0.032	0.026	0.047	
	0.048	0.029	0.042	0.043	0.044	0.034	0.038	0.035	0.032
3	0.049	0.035	0.031	0.035	0.065	0.035	0.038	0.036	0.045
	0.027	0.035	0.034	0.034	0.037	0.035	0.046	0.040	
	0.045	0.032	0.033	0.038	0.060	0.034	0.034	0.034	0.050
4	0.033	0.033	0.043	0.040	0.033	0.028	0.036	0.040	0.044
	0.037	0.050	0.034	0.034	0.039	0.044	0.038	0.035	
	0.034	0.031	0.035	0.044	0.047	0.037	0.043	0.038	0.042
5	0.037	0.033	0.032	0.035	0.037	0.036	0.045	0.032	0.029
	0.044	0.072	0.036	0.027	0.030	0.048	0.036	0.037	

对表 1.7 的结果进行分析，对每一个 i，寻找其 M_g 值接近于 0.065 的那一个值。这些具体的 g 决定了相应的移位 k_1、k_2、k_3、k_4、k_5。根据具体的数据，可以考虑密钥为 $k=(9,0,13,4,19)$，对应的字母为 JANET。尝试着解密密文，可以验证密钥是正确的。

解密后的明文为：

The almond tree was in tentative blossom. The days were longer, often ending with magnificent evenings of corrugated pink skies. The hunting season was over, with hounds

and guns put away for six months . The vineyards were busy again as well-organized farmers treated their vines and the more lackadaisical neighbors hurried to do the pruning they should have done in November.

1.6 Hill 密码系统

Hill 密码系统是一种比 Vigenère 密码系统更复杂的古典密码系统。

Hill 密码系统：在 Hill 密码系统中，明文空间和密文空间都是 Z_n^m，明文用 $x=(x_1, x_2, \cdots, x_m)$ 表示，密文用 $y=(y_1, y_2, \cdots, y_m)$ 表示。

密钥：

$$k = \begin{bmatrix} k_{11} & k_{12} & \cdots & k_{1m} \\ k_{21} & k_{22} & \cdots & k_{2m} \\ \vdots & \vdots & & \vdots \\ k_{m1} & k_{m2} & \cdots & k_{mm} \end{bmatrix}$$

实施加密算法：

$$(y_1, y_2, \cdots, y_m) = (x_1, x_2, \cdots, x_m) \begin{bmatrix} k_{11} & k_{12} & \cdots & k_{1m} \\ k_{21} & k_{22} & \cdots & k_{2m} \\ \vdots & \vdots & & \vdots \\ k_{m1} & k_{m2} & \cdots & k_{mm} \end{bmatrix}$$

即：

$$y = xk$$

相应的解密算法是：

$$x = yk^{-1}$$

例 1.12 Hill 密码系统中，$n=26$，$m=2$，明文是“sunday”，求其加解密过程。

将明文变成数字，并按每两个字母进行分组：

(18,20) (13,3) (0,24)

密钥：

$$k = \begin{bmatrix} 9 & 6 \\ 7 & 11 \end{bmatrix}$$

实施加密算法：

$$(y_1, y_2) = (x_1, x_2) \begin{bmatrix} 9 & 6 \\ 7 & 11 \end{bmatrix}$$

即：

$$\begin{cases} y_1 = 9x_1 + 7x_2 \\ y_2 = 6x_1 + 11x_2 \end{cases}$$

分别得到密文：

(16,16) (8,7) (12,4)

转化为英文密文是“qqihme”。

实施解密算法：

$$(x_1, x_2) = (y_1, y_2)\begin{bmatrix} 9 & 6 \\ 7 & 11 \end{bmatrix}^{-1} = (y_1, y_2)\begin{bmatrix} 23 & 4 \\ 9 & 7 \end{bmatrix}$$

得到明文：

$$(18,20) \quad (13,3) \quad (0,24)$$

即“sunday”。

在实施 Hill 解密算法时，需要计算 k^{-1}，二阶矩阵的逆可以按下列公式求得：

$$k^{-1} = (\det k)^{-1}\begin{bmatrix} k_{22} & -k_{12} \\ -k_{21} & k_{11} \end{bmatrix} (\bmod n)$$

其中：

$$\det k = k_{11}k_{22} - k_{12}k_{21} (\bmod n)$$

特别地，当 k 是对角矩阵时，Hill 密码系统可能退化成乘法密码系统；而当 k 是换位矩阵时，Hill 密码系统退化成换位密码系统。

1.7　流密码系统

虽然大部分古典密码系统的安全性都不是很好，但通过古典密码系统可以演化出一种在理论与实际中都无法破解的密码。这种绝对安全的密码是一次一密（One Time Padding）密码系统，由 Major Joseph Mauborgne 和 AT&T 公司的 Gilbert Vernam 于 1917 年发明。

为便于分析一次一密密码系统的安全性，首先明确以下几个概念。

计算安全（Computational Security）：如果攻击者破解密码的最好算法需要 N 次操作（N 是某个特定的足够大的数），则称这个密码系统是计算安全的。

在实际中，分析密码系统的安全性主要是分析它的计算安全性，而分析计算安全性，通常是将系统的计算安全性等价为一个“难解的数学问题”来研究。例如，如果某个密码系统的破解问题等价于离散对数求解问题（见后面的章节），那么该密码系统是计算安全的。

无条件安全（Unconditional Security）：如果攻击者利用无限的资源仍不能破解密码，则称这个密码系统是无条件安全的。

无条件安全又被称为理论安全。

完善保密性（Perfect Secrecy）：如果密文和明文在统计上相互独立，则称一个密码系统具有完善保密性。

完善保密性是从概率统计上来分析密码系统的安全性，而不是用确定性方法。

一次一密密码系统使用一次一密密码表，该密码表是一个不重复的随机密钥字母集合，在实际应用中将这个密钥字母集合写在几张纸上，并被粘成一个密码本。

一次一密密码系统通过电传打字机来传递信息。发送者用密码表中的每一密钥字母准确地加密一个明文字符。加密是明文字符和一次一密密码系统密钥字符的模 26 加法，且每个密钥仅对一个消息使用一次。发送者对所发送的消息加密后，销毁密码本中用过的一页或磁带部分。接收者有一个同样的密码本，并依次使用密码表上的每个密钥去解密密文的

每个字符。接收者在解密消息后销毁密码本中用过的一页或磁带部分。新的消息则用密码本中新的密钥加密。

如果窃听者不能得到用来加密消息的一次一密密码表,则该加密方案是完全保密的。给出的密文消息相当于同样长度的任何可能的明文消息。

由于每一密钥集合都以随机方式产生,因此各个字母出现的概率均等,攻击者没有任何信息用来对密文进行密码分析。

一次一密密码系统的加密方法可以很容易推广到二进制数据的加密,只需用由二进制数据组成的一次一密密码本代替由字母组成的一次一密密码序列,用异或代替一次一密密码系统的明文字符模加即可。为了解密,用同样的一次一密密码系统对密文异或,其他保持不变,保密性依然很完善。

例 1.13 假设明文消息是 011001110,密钥是 010100011 为密码本中最新没用到的部分,异或用⊕表示。

加密过程:

$$c = e_K(m) = 011001110 \oplus 010100011 = 001101101$$

解密过程:

$$m = d_K(c) = 001101101 \oplus 010100011 = 011001110$$

一次一密密码系统一直被认为是不可破解的,直到 1948 年这一命题才由 Shannon 提出的安全性理论证明其无条件安全性。对 Shannon 的基础理论感兴趣的读者,可阅读附录中的相关部分。

虽然一次一密密码系统是无条件安全的,但它并没有得到广泛应用。首先,由于密钥位必须是随机的,而且不能重复使用,这必然导致密钥序列的长度要等于消息的长度。这使得密钥的传输与存储都很困难,尤其对于加密较长的消息。其次,需要完全确保发送者和接收者的同步性,否则即使接收者有 1 位的偏移,消息都会变得完全面目全非。此外,生成一个完全随机数发生源也不是一件很容易的事情。

将一次一密密码系统的设计思想稍作改动可以形成相对实用的流密码系统。

与一次一密密码系统的密钥产生机制不同,流密码系统的密钥不是完全随机的。加密和解密所需要的这种序列由一个确定性的密钥流生成器(Key Generator)产生,该生成器的输入是一个容易记住的密钥,称为密钥流生成器的初始密钥或种子(Seed)密钥。从这个意义上来说,密钥流序列是伪随机序列(Pseudorandom Sequence)。

完整的流密码系统模型,如图 1.1 所示。

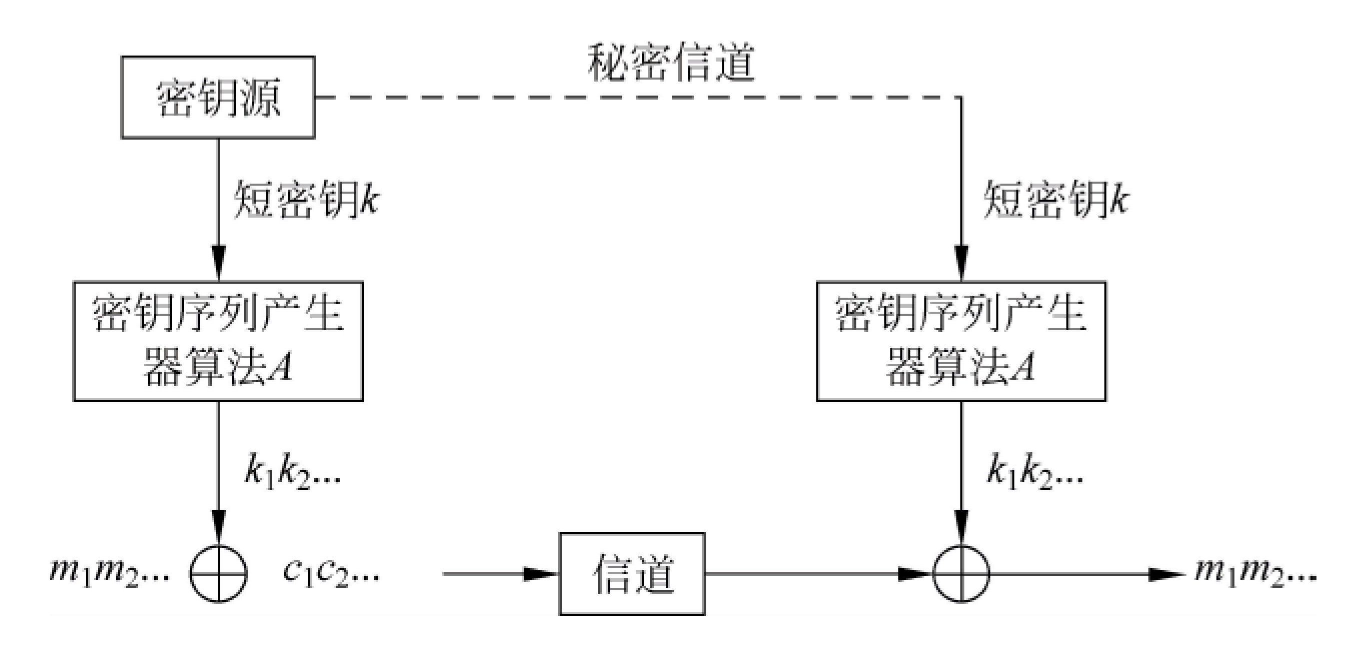


图 1.1 流密码系统模型

从图 1.1 的模型可以看出，流密码体制的安全强度完全取决于密钥流的安全性。因此，什么样的伪随机序列是安全可靠的密钥流序列，如何构造这种序列就是流密码研究中的关键问题。

根据密码流与明文的关系，可将流密码进一步划分成同步流密码和自同步流密码两类。

在同步流密码中，密钥流的产生与明文消息流相互独立。

由于密钥流与明文串无关，所以同步流密码中的每个密文 c_i 不依赖于之前的明文 m_{i-1}、…、m_1。同步流密码的一个重要优点就是无错误传播：在传输期间一个密文字符被改变只影响该符号的恢复，不会对后继的符号产生影响。

但是，在同步流密码中发送方和接收方必须是同步的，用同样的密钥且该密钥操作在同样的位置时才能保证正确解密。如果在传输过程中密文字符有插入或删除导致同步丢失，密文与密钥流将不能对齐，导致无法正确解密。要正确还原明文，密钥流必须再次同步。

在自同步流密码中，密钥流的产生与之前已经产生的若干密文（或明文）有关，其加密过程，如图 1.2 所示。

图 1.2　自同步流密码系统的加密过程

下面介绍的自动密钥密码系统正是基于这种思想。

自动密钥密码系统（Autokey Cipher）：明文空间、密文空间、密钥空间分别用 P、C、K 表示，$P=C=K=Z_{26}$，令 $z_1=K$ 且：

$$z_i = x_{i-1}(i \geqslant 2)$$

x、y、$z \in Z_{26}$，x 表示明文，y 表示密文。

加密算法：

$$e_z(x) = x + z \bmod 26$$

解密算法：

$$d_z(y) = y - z \bmod 26$$

例 1.14　设密钥 $k = 8$，明文是"RENDEZVOUS"，求其加解密过程。

首先将明文转换成整数串，得到：

17 4 13 3 4 25 21 14 20 18

密钥流是：

8 17 4 13 3 4 25 21 14 20

用密钥流对明文进行加密，加密是模加运算，可得加密结果：

25 21 17 16 7 3 20 9 8 12

写成字母的形式，文本方式的密文是：

ZVRQHDUJIM

接下来进行解密。先将密文写成整数串的形式：

25 21 17 16 7 3 20 9 8 12

由于密钥 $k=8$，可将第一位 25 取出，计算：

$$x_1 = 25 - 8 \bmod 26 = 17$$

然后，根据 x_1 的计算结果，计算：

$$x_2 = 21 - 17 \bmod 26 = 4$$

类似地，可以计算出：

$$x_3 = 17 - 4 \bmod 26 = 13$$
$$x_4 = 16 - 13 \bmod 26 = 3$$
$$x_5 = 7 - 3 \bmod 26 = 4$$
$$x_6 = 3 - 4 \bmod 26 = 25$$
$$x_7 = 20 - 25 \bmod 26 = 21$$
$$x_8 = 9 - 21 \bmod 26 = 14$$
$$x_8 = 8 - 14 \bmod 26 = 20$$
$$x_8 = 12 - 20 \bmod 26 = 18$$

得到明文：

17 4 13 3 4 25 21 14 20 18

即"RENDEZVOUS"，解密成功。

自动密钥密码系统的安全性并不高，因为仅仅有 26 个可能的密码，很容易被猜出。但是，这并不意味着流密码系统的安全性不高，相反随着电子技术的发展，线性反馈移位寄存器(LFSR)被引入到流密码系统来，使得流密码系统成为风靡一时的主流密码系统。基于 LFSR 的流密码系统将在第 2 章详细介绍，因为基于 LFSR 的流密码系统带有太多的"科学"特征，与古典密码系统的距离已渐行渐远。

对密码学历史感兴趣的读者还可以阅读戴维·卡恩(David Kahn)的作品《破译者》(The Code Breakers)，书中详细记述了从三千年前直到第二次世界大战时期密码学的发展历程。

1.8 习　　题

1. 设 $k=3$，请使用移位密码体制对下面的消息进行加密：

change password is a request reply protocol that includes a krbpriv message that contains the new password for the user the original change password protocol does not allow an administrator to set a password for a new user this functionality is useful in some environments and this proposal extends the change password protocol to allow password setting the changes are adding new fields to the request message to indicate the principal which is having its password set not requiring the initial flag in the service ticket using a new protocol version number and adding three new result codes

2. 在移位密码系统中，密文如下：

estd oznfxpye opdnctmpd esp xo xpddlrp otrpde lwrzctesx esp lwrzctesx elvpd ld tyafe l xpddlrp zq lcmteclcj wpyres lyo aczofnpd ld zfeafe l i mte qtyrpcactye zc xpddlrp otrpde zq esp tyafe te td nzyupnefcpo esle te td nzxafeletzylwwj tyqpldtmwp ez aczofnp ehz xpddlrpd slgtyr esp dlxp xpddlrp otrpde zc ez aczofnp lyj xpddlrp slgtyr l rtgpy acpdapntqtpo elcrpe

xpddlrp otrpde esp xo lwrzctesx td tyepyopo qzc otrtelw dtrylefcp laawtnletzyd hspcp l wlcrp qtwp xfde mp nzxacpddpo ty l dpnfcp xlyypc mpqzcp mptyr pyncjaepo htes l actglep vpj fyopc l afmwtn vpj ncjaezdjdepx dfns ld cdl
试破解该密码系统。

3. 使用仿射密码算法对信息“take a clean pair of heels”进行加密，已知 $k_0=7$，$k_1=11$。

4. 使用仿射密码算法对信息“hlzb l slqfb tpnywbotalojb”进行解密，已知 $k_0=11$，$k_1=17$。

5. 求 delta$\begin{bmatrix} 13 & 7 \\ 9 & 17 \end{bmatrix}$ mod 26。

6. 求 99^{-1} mod 2003。

7. 求 24770376 和 50572851 的最大公约数。

8. 使用 Vigenère 密码算法对信息“take a bear by the tooth”进行加密，已知密钥 $k=\{9,22,15,2\}$。

9. 使用 Vigenère 密码算法对信息“dlsh r nwqjaqflzcv wtoxip”进行解密，已知密钥 $k=\{17,11,8,3\}$。

10. 使用 Hill 密码算法对信息“have a button missing”进行加密，已知密钥 $k=\begin{bmatrix} 19 & 5 \\ 9 & 10 \end{bmatrix}$。

11. 求$\begin{bmatrix} 21 & 4 \\ 5 & 8 \end{bmatrix}^{-1}$ mod 26。

12. 使用 Hill 密码算法对信息“mheyev”进行解密，已知密钥 $k=\begin{bmatrix} 16 & 3 \\ 11 & 12 \end{bmatrix}$。

13. 在自动密钥密码系统中，已知密钥 $k=9$，明文是“expertssaytargetedlegislationleads tostrongdemocracies”，求密文。

第 2 章　对称密码系统

对称密码系统的概念并不陌生，因为第 1 章中介绍的古典密码系统基本上都是对称密码系统。在一个对称密码系统中，加密与解密过程中的密钥是一样的，或者可以相互推导得出。相对于古典密码系统，本章中将要介绍的对称密码系统更加复杂，基于更多的数学原理和更强的电子计算机技术。

本章重点介绍 3 种对称密码系统：基于 LFSR 的流密码系统、DES 算法和 AES 算法。此外，还将介绍主流的加密模式，加密模式对于对称密码系统的实现尤为重要。

2.1　基于 LFSR 的流密码系统

第 1 章介绍了自动密钥密码系统，该系统的密钥只有 26 个，安全性并不高。由于流密码的安全强度取决于密钥流生成器生成的密钥流，所以如何生成安全性好的密钥流便成了流密码系统的焦点。

接下来将介绍一种基于线性反馈移位寄存器的流密码系统。线性反馈移位寄存器的方法具有以下优点：

（1）LFSR 的结构非常适合硬件实现；

（2）LFSR 的结构便于使用代数方法进行分析；

（3）产生序列的周期可以很大；

（4）产生的序列具有较好的统计特性。

反馈移位寄存器，如图 2.1 所示。

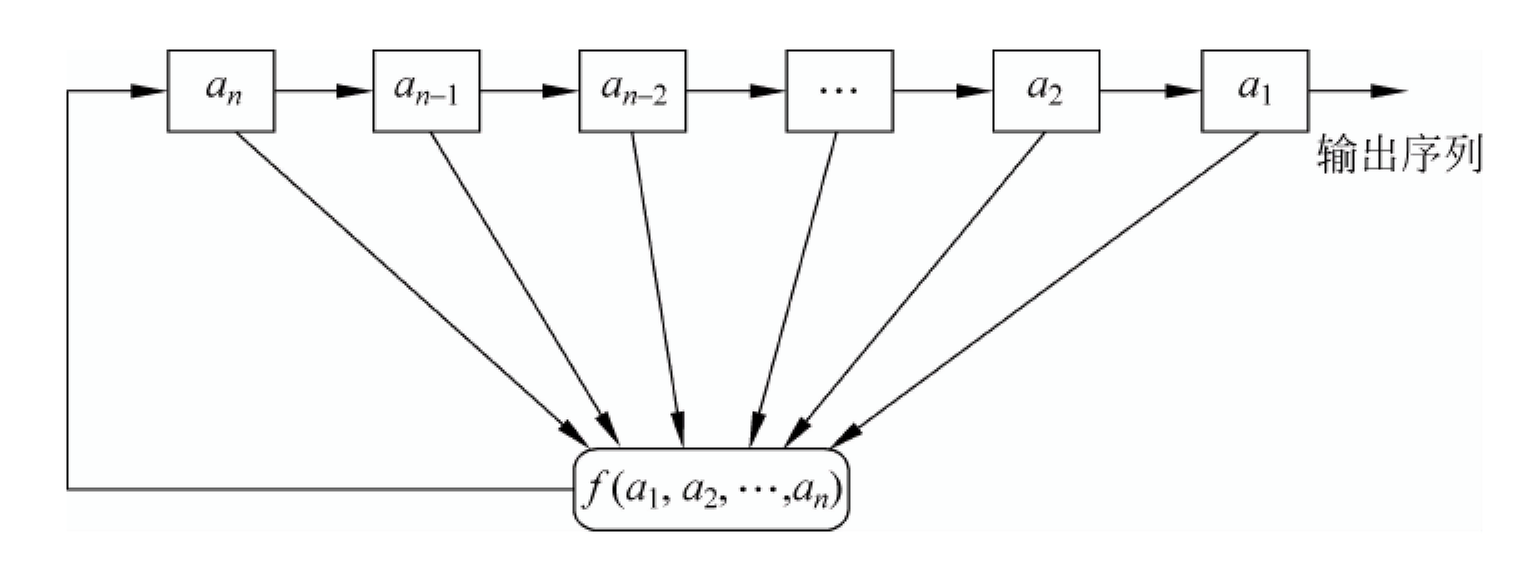


图 2.1　反馈移位寄存器

在图 2.1 中，a_i 表示存储单元，取值为 0 或 1，a_i 的个数 n 称为反馈移位寄存器的级。

在某一时刻，这些级的内容构成该反馈移位寄存器的一个状态，总共有 2^n 个可能的状态，每一个状态对应于二进制的一个 n 维向量，用 $(a_1, a_2, \cdots, a_n)$ 表示。

函数 f 是一个 n 元布尔函数，称为反馈函数。

在每个周期，各级存储器 a_i 将其存储内容向下一级 a_{i-1} 传递，最右边一级存储器的内

容作为该时刻的输出。

根据该时刻寄存器的状态计算出 $f(a_1,a_2,\cdots,a_n)$ 并把该值作为最左边一级寄存器下一时刻的内容。

显然，一个反馈移位寄存器的逻辑功能完全由该移位寄存器的反馈函数所决定。

反馈函数形如：

$$f(a_1,a_2,\cdots,a_n)=c_na_1\oplus c_{n-1}a_2\oplus\cdots\oplus c_1a_n$$

其中，系数 $c_i\in\{0,1\}$，这里的运算是二进制的模加和模乘。由于该反馈函数是 a_1、a_2、$\cdots$、a_n 的线性函数，因此对应的反馈移位寄存器称为线性反馈移位寄存器(Linear Feedback Shift Register，LFSR)，如图 2.2 所示。

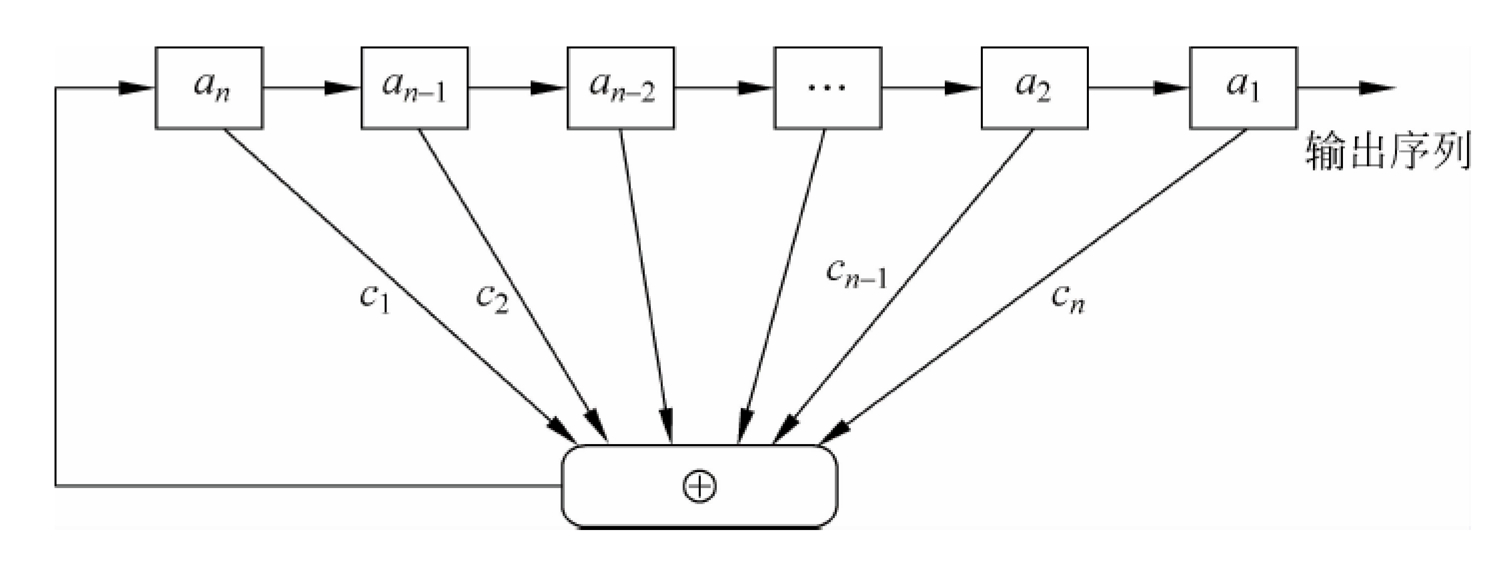


图 2.2 线性反馈移位寄存器

根据 LFSR 中反馈函数的系数 $c_i(i=1,2,\cdots,n)$ 取值的不同，这样的反馈函数应该有 2^n 种。令 $a_i(t)$ 表示 t 时刻第 i 级寄存器的内容，则第 $t+1$ 时刻寄存器的内容为：

$$a_i(t+1)=a_{i+1}(t),\quad i=(1,2,\cdots,n-1)$$

$$a_n(t+1)=c_na_1(t)\oplus c_{n-1}a_2(t)\oplus\cdots\oplus c_1a_n(t)$$

称多项式

$$c_nx^n+c_{n-1}x^{n-1}+\cdots+c_1x+1$$

为该 LFSR 的联接多项式。

显然，LFSR 的联接多项式与反馈函数一一对应。如果知道 LFSR 的反馈函数，就可以求出 LFSR 的联接多项式；反之亦然。

例 2.1 以 4 级线性反馈移位寄存器为例，如图 2.3 所示，设状态转移关系为：

$$a_i(t+1)=a_{i+1}(t),\quad i=1,2,3$$

$$a_4(t+1)=a_1(t)\oplus a_4(t)$$

图 2.3 4 级 LFSR 的结构

假设初始状态为$(a_1, a_2, a_3, a_4)=(0,1,1,0)$,则可根据反馈函数计算出该 LFSR 在各时刻的所有状态,如表 2.1 所示。

表 2.1 4 级 LFSR 的全状态表

t	a_4	a_3	a_2	a_1	t	a_4	a_3	a_2	a_1
0	0	1	1	0	8	1	1	1	0
1	0	0	1	1	9	1	1	1	1
2	1	0	0	1	10	0	1	1	1
3	0	1	0	0	11	1	0	1	1
4	0	0	1	0	12	0	1	0	1
5	0	0	0	1	13	1	0	1	0
6	1	0	0	0	14	1	1	0	1
7	1	1	0	0	15	0	1	1	0

由状态转移过程可见,当 $t=15$ 时该寄存器的状态恢复至 $t=0$ 时刻的状态,因此之后的状态将开始周期性重复。

这个移位寄存器输出的序列就是:

$$011001000111101\ 011001000111101\ \cdots$$

序列的周期为 $15(2^4-1)$。

还可以用更为直观的状态转移图来描述状态转移过程,如图 2.4 所示。

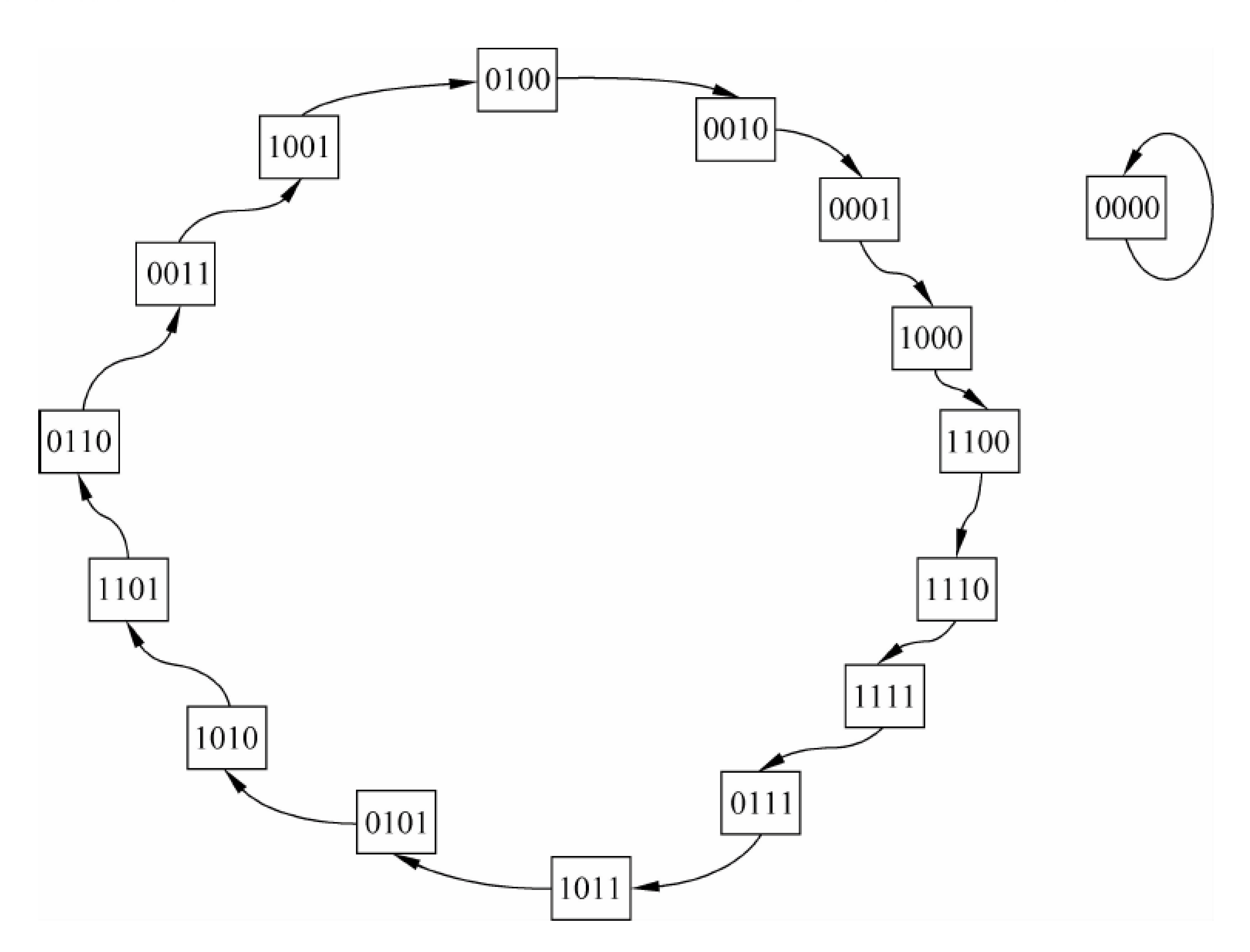


图 2.4 4 级 LFSR 的状态转移图

该 LFSR 的状态转移图仅由两个圈构成,其中一个是由全零状态构成的长度为 1 的圈,另一个是由全部其余 2^n-1 个状态构成的长度为 2^n-1 的圈。

并非所有的 4 级 LFSR 的状态转移图都是这样的形式,如图 2.5 所示的示例就是一个特例。

例 2.2　在 4 级 LFSR 中,假设联接多项式为 $x^4+x^3+x^2+x+1$。

初始状态 0001 对应的状态转换图如图 2.6 所示。

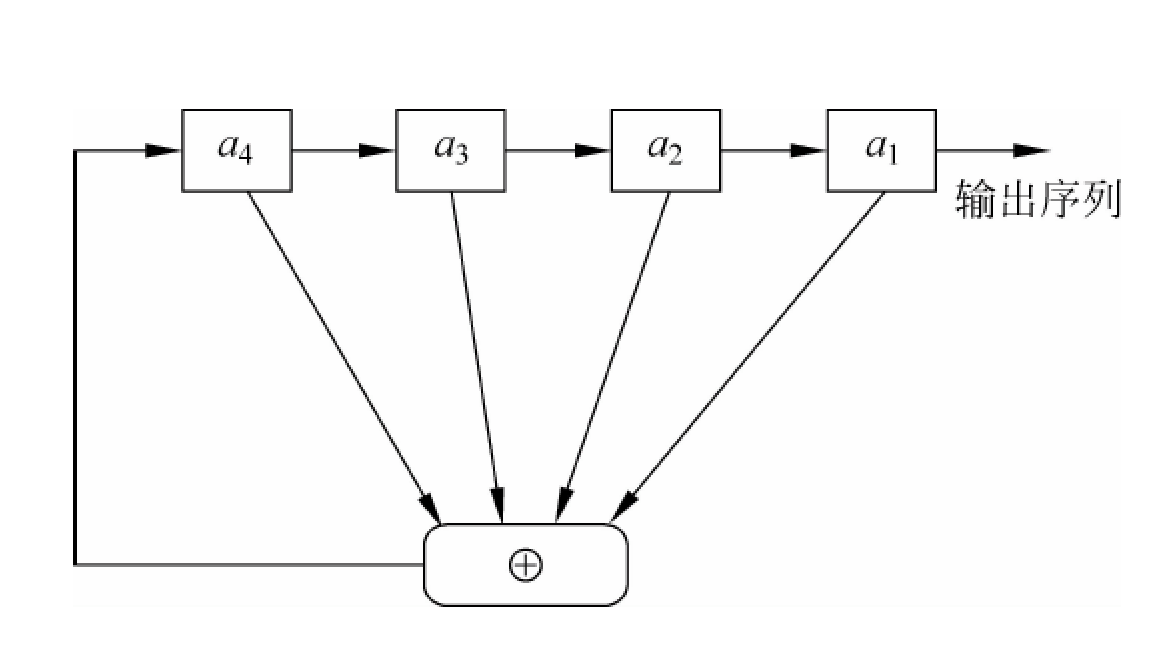


图 2.5　4 级 LFSR 的结构示例

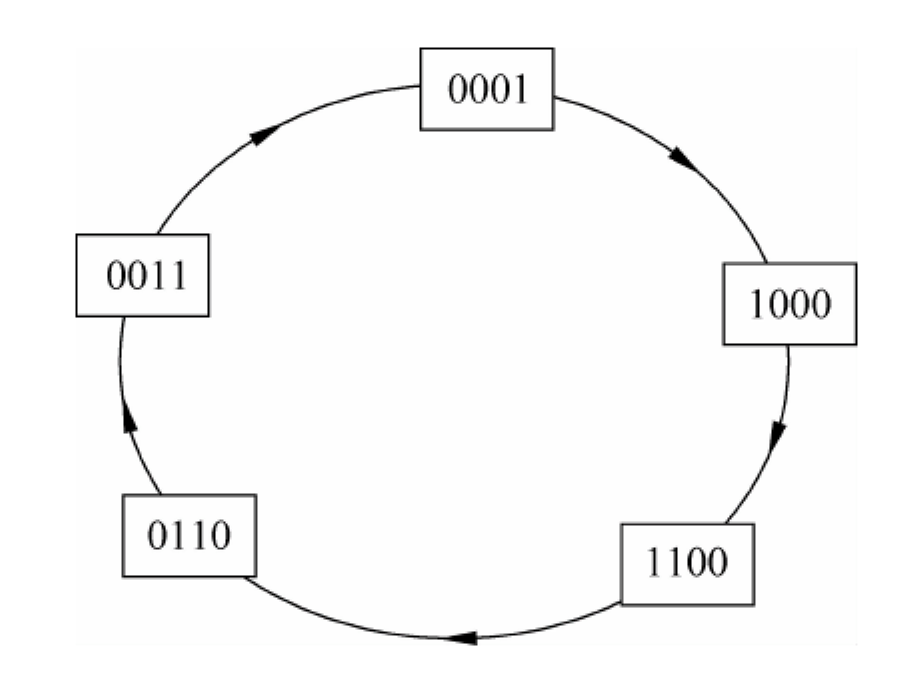


图 2.6　初始状态 0001 对应的状态转换图

初始状态 0010 对应的状态转换图如图 2.7 所示。

初始状态 1111 对应的状态转换图如图 2.8 所示。

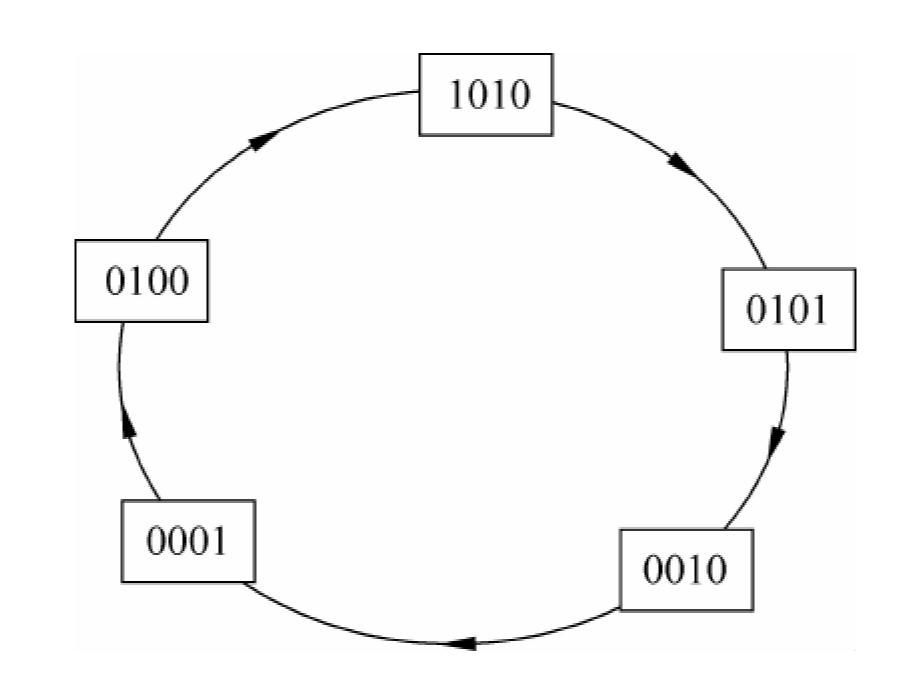


图 2.7　初始状态 0010 对应的状态转换图

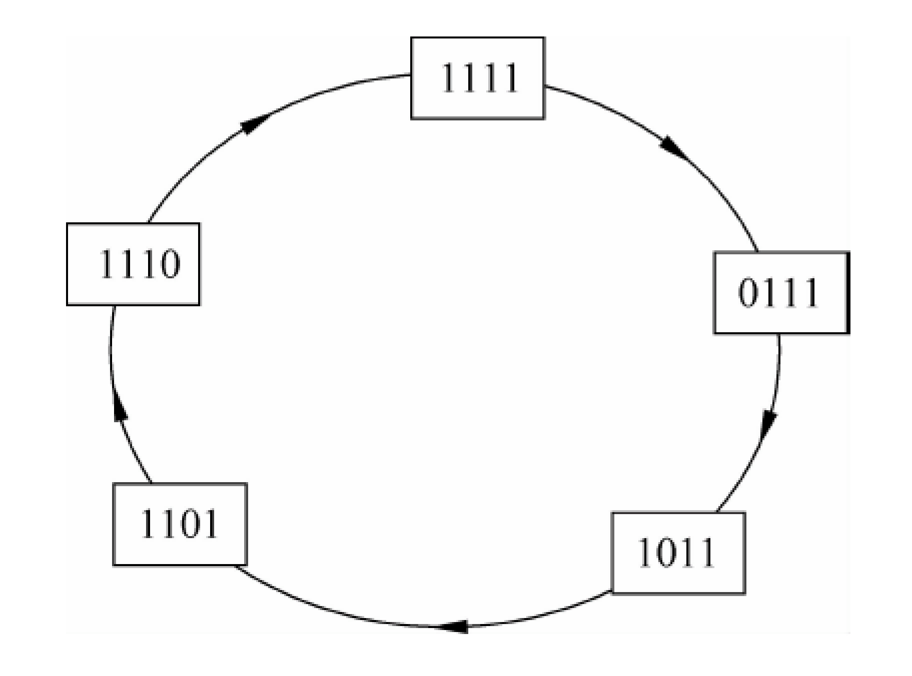


图 2.8　初始状态 1111 对应的状态转换图

以上 3 个状态转换图中的状态分别是 5 个,加上 0000,构成全部的 16 种状态。

在实际应用中,状态转换周期的大小会影响流密码系统的安全性,往往希望找到周期尽量大的循环序列。

***n* 级最大周期线性移位寄存器序列**:如果二进制上的 n 次多项式 $c_nx^n+c_{n-1}x^{n-1}+\cdots+c_1x+1$ 为联接多项式的 n 级 LFSR 产生的非零序列周期为 2^n-1,称这个序列是 n 级最大周期线性移位寄存器序列,简称 m 序列。产生周期为 2^n-1 的 m 序列的 n 级 LFSR 又称最长线性移位寄存器。

本原多项式:n 次多项式 $f(x)$,如果满足下列条件:

(1) $f(x)$为不可约多项式;

(2) $f(x)$可整除 x^m+1,$m=2^n-1$;

(3) $f(x)$不能整除 x^q+1,$q<m=2^n-1$;

那么称 $f(x)$为本原多项式。

不难看出,一个序列是否是 m 序列应当与产生这一序列的 LFSR 的联接多项式密切相关。这里不加证明地给出以下结论:

n 级 LFSR 所产生的非零序列是 m 序列的充要条件，是其联接多项式为二进制上的本原多项式。

这一结论同时也表明：一个 LFSR 为最长移位寄存器的充要条件是其联接多项式为本原多项式。

表 2.2 列出了 $n \leqslant 10$ 的常用本原多项式。

表 2.2 $F(x)$ 上常用的本原多项式

n	本原多项式	n	本原多项式
2	x^2+x+1	7	x^7+x^3+1
3	x^3+x+1	8	$x^8+x^4+x^3+x^2+1$
4	x^4+x+1	9	x^9+x^4+1
5	x^5+x^2+1	10	$x^{10}+x^3+1$
6	x^6+x+1		

2.2 DES

1973 年，美国标准和技术协会（NIST）开始着手研究除国防部外的其他部门的计算机系统的数据加密标准，于 1973 年 5 月 15 日和 1974 年 8 月 27 日先后两次向公众发出了征求加密算法的公告。加密算法的要求包括以下几点：

(1) 提供高质量的数据保护，防止数据未经授权的泄露和未被察觉的修改。

(2) 具有相当高的复杂性，使得破译的开销超过可能获得的利益，同时又要便于理解。

(3) DES 密码体制的安全性应该不依赖于算法的保密，其安全性仅以加密密钥的保密为基础。

(4) 实现经济，运行有效，并且适用于多种完全不同的应用。

经过多轮筛选，1977 年 1 月美国政府声明：采纳 IBM 公司设计的方案作为非机密数据的正式数据加密标准（Data Encryption Standard，DES）。DES 密码体制基于早期的 Lucifer 密码，是一种典型的分组密码体制。

DES 算法具有对称性，既可用于加密又可用于解密。对称性带来的一个很大的好处在于硬件实现，DES 的加密和解密可以用完全相同的器件来实现。

DES 算法的明文分组是 64 位，输出密文也是 64 位。所用密钥的有效位数是 56 位，加上校验位共 64 位。

总体流程如表 2.3 所示。输入的 64 位明文，先经初始 IP 变换，形成 64 位数据，64 位数据被分成两部分，分别是 L 部分和 R 部分；L 和 R 经 16 次迭代，形成新的 64 位数据；新的 64 位数据再经初始逆变换，输出 64 位密文。

下面分步进行描述。

表 2.3　DES 算法流程

输入 64 位数据	
初始变换(IP)	
$L(0)$	$R(0)$
$L(1)=R(0)$	$R(1)=L(0)$ xor $F(R(0),K(1))$
$L(2)=R(1)$	$R(2)=L(1)$ xor $F(R(1),K(2))$
…	
$L(15)=R(14)$	$R(15)=L(14)$ xor $F(R(14),K(15))$
$L(16)=R(15)$	$R(16)=L(15)$ xor $F(R(15),K(16))$
$L(17)=R(16)$	$R(17)=L(16)$
初始逆变换(IP^{-1})	
输出 64 位数据	

1. 初始变换 IP

IP 初始变换由一个 8×8 的变换矩阵来完成，如表 2.4 所示。初始变换是线性变换，它使明文发生位置上的变化。

设 x 是明文，变换后 $x_0=\mathrm{IP}(x)=L_0R_0$，这里 L_0 和 R_0 都是 32 位。

表 2.4　IP 初始变换表

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

2. 迭代运算

迭代运算按以下规则进行：

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$$

其中，$1\leqslant i\leqslant 16$，即迭代次数是 16 次。

L_i 和 R_i 都已确定，下面来看函数 $f(R,k)$。

函数 $f(R,k)$中第一个变量是 32 位，第二个变量是 48 位子密钥(子密钥的产生将在后面具体介绍)，输出是 32 位，如表 2.5 所示。

表 2.5　$F(R(i),K(i+1))$

$r_{32}(i)$ xor $k_1(i+1)$	$r_1(i)$ xor $k_2(i+1)$	…	$r_5(i)$ xor $k_6(i+1)$
$r_4(i)$ xor $k_7(i+1)$	$r_5(i)$ xor $k_8(i+1)$	…	$r_9(i)$ xor $k_{12}(i+1)$
⋮	⋮		⋮
$r_{28}(i)$ xor $k_{43}(i+1)$	$r_{29}(i)$ xor $k_{44}(i+1)$	…	$r_1(i)$ xor $k_{48}(i+1)$

迭代运算的具体过程如下：

（1）将第一个变量 R 经一个扩充函数 E 扩充成 48 位的串，扩充函数如表 2.6 所示。

表 2.6 扩充函数

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

（2）计算 $E(R)\oplus k$，并将所得结果分成 8 个长度为 6 的串，即 $T_1T_2\cdots T_8$。

（3）下面对 S 盒进行介绍。

每个 S 盒是一个固定的 4×16 矩阵，它的元素由 0～15 组成，0～15 出现的顺序不同，如表 2.7 所示。

T_i 由 $t_1t_2\cdots t_6$ 构成，$S_i(T_i)$是这样一个非线性运算：用 t_1t_6 对应的整数来确定 S_i 的行，用 $t_2t_3t_4t_5$ 对应的整数来表确定 S_i 的列，S_i 在该行该列对应的二进制表示就是 $S_i(T_i)$的取值。

表 2.7 *S* 替换盒

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S(1)	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S(2)	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S(3)	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S(4)	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S(5)	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

续表

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S(6)	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S(7)	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S(8)	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

例 2.3　$S_1(101110)$。

将 101110 按 1、6 和 2、3、4、5 进行组合，得到 10 和 0111，转化为十进制分别是 2 和 7。查询 $S(1)$替代盒，从中找出第 2 行、第 7 列（注意：S 盒中的行号和列号都是从 0 开始），相应的值是 11。于是可得：

$$S_1(101110) = 1011$$

(4) 每一组 6 位串 T_i 在通过 S 盒后将变成 4 位串，48 位串在通过 8 个 S 盒后将形成 32 位的串，记为 $C = C_1C_2\cdots C_{32}$，C 再经过一个换位变换 P，得到 $f(R,k)$，如表 2.8 所示。

表 2.8　$f(R(i),K(i+1))$中的换位变换

16	7	20	21	2	8	24	14
29	12	28	17	32	27	3	9
1	15	23	26	19	13	30	6
5	18	31	10	22	11	4	25

3. 初始变换的逆变换 IP^{-1}

表 2.9 是初始变换 IP 的逆变换。

表 2.9　初始变换的逆变换 IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	52	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

接下来介绍 DES 子密码的生成，表 2.10 描述了子密码的生成流程。

表 2.10　子密码生成流程

原始 64 位密钥		
经换位选择 1 得 56 位密钥		
$C(0)$(28 位)	$D(0)$(28 位)	
$C(1)=\lambda\sigma(1)C(0)$(左移位)	$D(1)=\lambda\sigma(1)D(0)$(左移位)	拼接 $C(1)D(1)$,经换位选择 2 得 $K(1)$
$C(2)=\lambda\sigma(2)C(1)$	$D(2)=\lambda\sigma(2)D(1)$	$K(2)$
⋮		
$C(15)=\lambda\sigma(15)C(14)$	$D(15)=\lambda\sigma(15)D(14)$	$K(15)$
$C(16)=\lambda\sigma(16)C(15)$	$D(16)=\lambda\sigma(16)D(15)$	$K(16)$

原始的 64 位密钥并不是全部有效,在位置 8、16、…、64 上的数据是校验位,因此有效密钥长度实际上只有 56 位。

初始密钥首先经过换位选择,换位选择矩阵如表 2.11 所示。

表 2.11　换位选择 1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

换位选择 1 的矩阵是 8×7 的矩阵,正好是 56 位,没有包含 8、16、…、64 等,实质上是将校验位排除在外。

经换位选择 1 后的 56 位串经过 16 次迭代,每次迭代产生一个 64 位子密钥。

迭代过程主要是移位处理和换位选择 2 处理。

移位函数 $\lambda\sigma(i)$是一个非线性函数,如表 2.12 所示。

表 2.12　移位函数

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\sigma(i)$	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

换位函数 2 是一个 8×6 的矩阵,如表 2.13 所示。

表 2.13　换位函数 2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

DES 算法的加密子密钥和解密子密钥是相反的序列，即 $kd(i)=ke(16-i)$。

例 2.4 DES 算法实例。

64 位明文的十六进制表示为：

“6101035456016789”

相应的二进制表示为：

“0110000100000001000000110101010001010110000000010110011110001001”

56 位的密钥的二进制表示为：

“01000010011010011101101111111010101110111011010011011100”

加上奇校验后的 64 位密钥 Key 二进制表示为：

“0100001100110100011101100111111110101011110111001101001110111001”

相应的十六进制表示为：

“4334767fabdcd3b9”

加密得到密文：

$L(16)$ = “11100111111010100110011100001101”

$R(16)$ = “01111011011001010111100110101011”

DES 算法颁布之后，引起了学术界和企业界的广泛重视。众多厂商开始生产实现 DES 算法的硬件产品，一些公司在市场上买到高效率的 DES 硬件产品后，开始对重要数据进行加密，从而大大推动了密码技术的发展。

与此同时，在对 DES 密码进行了深入的研究后，学术界和企业界围绕它的安全性展开了激烈的争论。有趣的是，DES 的怀疑者和批评者并不比拥护者少。

从技术上说，对 DES 的批评主要集中在以下 3 个方面：

(1) 作为分组密码，DES 的加密单位仅有 64 位二进制，这对于数据传输来说太小，因为每个分组仅含 8 个字符。

(2) 密钥也只有 56 位二进制未免太短，各次迭代中使用的密钥是递推产生的，这种相关必然降低密码体制的安全性。

(3) 实现替代函数 S_i 所用的 S 盒的设计原理尚未公开，可能留有某种“后门”，知道秘密的人或许可以轻易地破解密文。

针对以上 DES 的缺陷，研究者提出了几种增强 DES 安全性的方法，主要包括以下 3 种：

(1) 三重 DES 算法。用 3 个不同密钥的三重加密：

$$C = E_{k_3}(D_{k_2}(E_{k_1}(P)))$$

$$P = D_{k_1}(E_{k_2}(D_{k_3}(C)))$$

这种方法由密码专家 Merkle 和 Hellman 推荐。

(2) 具有独立子密钥的 DES 算法。每一轮迭代都使用一个不同的子密钥，而不是由一个 56 位二进制的密钥产生。由于 16 轮迭代的每一轮使用一个 48 位二进制的密钥，所以该方法通过降低子密钥的相关性，增强了 DES 的加密强度。

(3) 使用交换 S 盒的 DES 算法。Biham 及 Shamir 证明通过优化 S 盒的设计或变换 S

盒本身的顺序，可以增强 DES 算法的加密强度。

2.3 AES

1997 年，NIST 发起征集高级加密标准(Advanced Encryption Standard，AES)算法的活动，并成立 AES 工作组，希望寻找一种新的分组密码算法以取代 DES 算法。征集 AES 的规则包括以下几点：

(1) AES 的设计应使其密钥长度可根据需要增加；

(2) AES 应易于在硬件和软件中实施的；

(3) AES 应免费提供，或在符合 ANSI 的专利政策条件下提供。

除需要满足以上基本条件外，算法的优劣将在以下几个方面比较：

(1) 安全性(对抗密码分析)；

(2) 计算效率；

(3) 内存的要求；

(4) 硬件和软件的适用性。

1998 年，NIST 召开 AES 候选算法会议，公布了 15 个 AES 候选算法。

1999 年 3 月 22 日，举行了第二次 AES 候选会议，从候选算法中选出 5 个，入选 AES 的 5 种算法分别是 MARS、RC6、Serpent、Twofish 和 Rijndael 算法。

2000 年 10 月，美国商务部部长 Norman 宣布：经过三年来世界各著名密码专家之间的竞争，“Rijndael 数据加密算法”最终获胜。2001 年，AES 算法正式公布。

AES 采用了两位比利时密码学家 Joan daemen 和 Vincent Rijmen 的密码算法方案，称为 Rijndael 算法。

类似于 DES，AES 也是一种迭代型的分组密码。迭代的轮数 N_r 由明文的分组长度 N_b 和密钥的长度 N_k(128/192/256 位)共同决定。AES 的明文分组长度 N_b 是 128 位，也就是 4 个字节，N_k 对应 128、192 和 256 位，转换为字节表示分别是 4、6 和 8。因此针对不同的密钥长度的循环轮数，如表 2.14 所示。

表 2.14 循环轮数 N_r 的取值

N_r	$N_b=4$	$N_b=6$	$N_b=8$
$N_k=4$	10	12	14
$N_k=6$	12	12	14
$N_k=8$	14	14	14

在加密之前，需要对明文数据块做预处理。

首先把明文块写成字的形式，每个字包含 4 个字节，同时每个字节是 $GF(2^8)$ 域的元素。其次把字节记为列的形式。类似地，密钥也要做相应的处理。例如，$N_k=4$ 的密钥可以记为表 2.15 的形式。

表 2.15　$N_k=4$ 的密钥表示形式

$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$

AES 运算的对象是字节，而字节运算是 GF(2^8)域中的运算。

下面对 GF(2^8)域进行简单的描述。

GF(2^8)域由从$(00)_{16}$到 $(FF)_{16}$的数值集合以及定义在该集合中加法和乘法运算构成。

GF(2^8)域的元素一般有两种表现形式，分别是数值和多项式表现形式。

例 2.5　字节 $B=(57)_{16}$可以表示为二进制形式：

$$B = b_7b_6b_5b_4b_3b_2b_1b_0 = 01010111$$

多项式形式是将一个由 $b_7b_6b_5b_4b_3b_2b_1b_0$ 组成的字节 B 表示为系数为{0,1}的二进制多项式，即：

$$b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0$$

因此，$(57)_{16}$表示成多项式的形式是：

$$x^6+x^4+x^2+x+1$$

GF(2^8)域中的主要运算有加法和乘法。两个多项式的加法定义为相同指数项的系数之和再模 2，简单地说，就是按位进行 XOR 运算。由于每个元素的加法逆元等于自己本身，所以减法运算和加法结果相同。

例 2.6　求$(57)_{16}+(83)_{16}$。

$$\begin{aligned}&(57)_{16}+(83)_{16}\\&=(01010111)_2+(10000011)_2\\&=(11010100)_2\\&=(D4)_{16}\end{aligned}$$

或采用其多项式记为：

$$(x^6+x^4+x^2+x+1)+(x^7+x+1)=x^7+x^6+x^4+x^2$$

将字节和多项式联系起来的方法同样也适用于 GF(2^8) 域中的乘法。在乘法运算中，需要把两个多项式相乘的结果，再对一个不可约多项式 $m(x)$取模。在 AES 中，这个不可约多项式约定为：

$$m(x)=x^8+x^4+x^3+x+1 \quad 或 \quad (11B)_{16}$$

例 2.7　求$(57)_{16}\cdot(83)_{16}$。

$$\begin{aligned}&(57)_{16}\cdot(83)_{16}\\&=(x^6+x^4+x^2+x+1)\cdot(x^7+x+1)\\&=x^{13}+x^{11}+x^9+x^8+x^7+x^7+x^5+x^3+x^2+x+x^6+x^4+x^2+x+1\\&=(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1)\ \mathrm{mod}\ (x^8+x^4+x^3+x+1)\\&=x^7+x^6+1\\&=(C1)_{16}\end{aligned}$$

AES 的加密流程如图 2.9 所示，其流程包括字节代换(SubBytes)、行移变换(ShiftRows)、混合列变换(MixColumns)和扩展密钥(AddRoundKey)4 个关键操作。

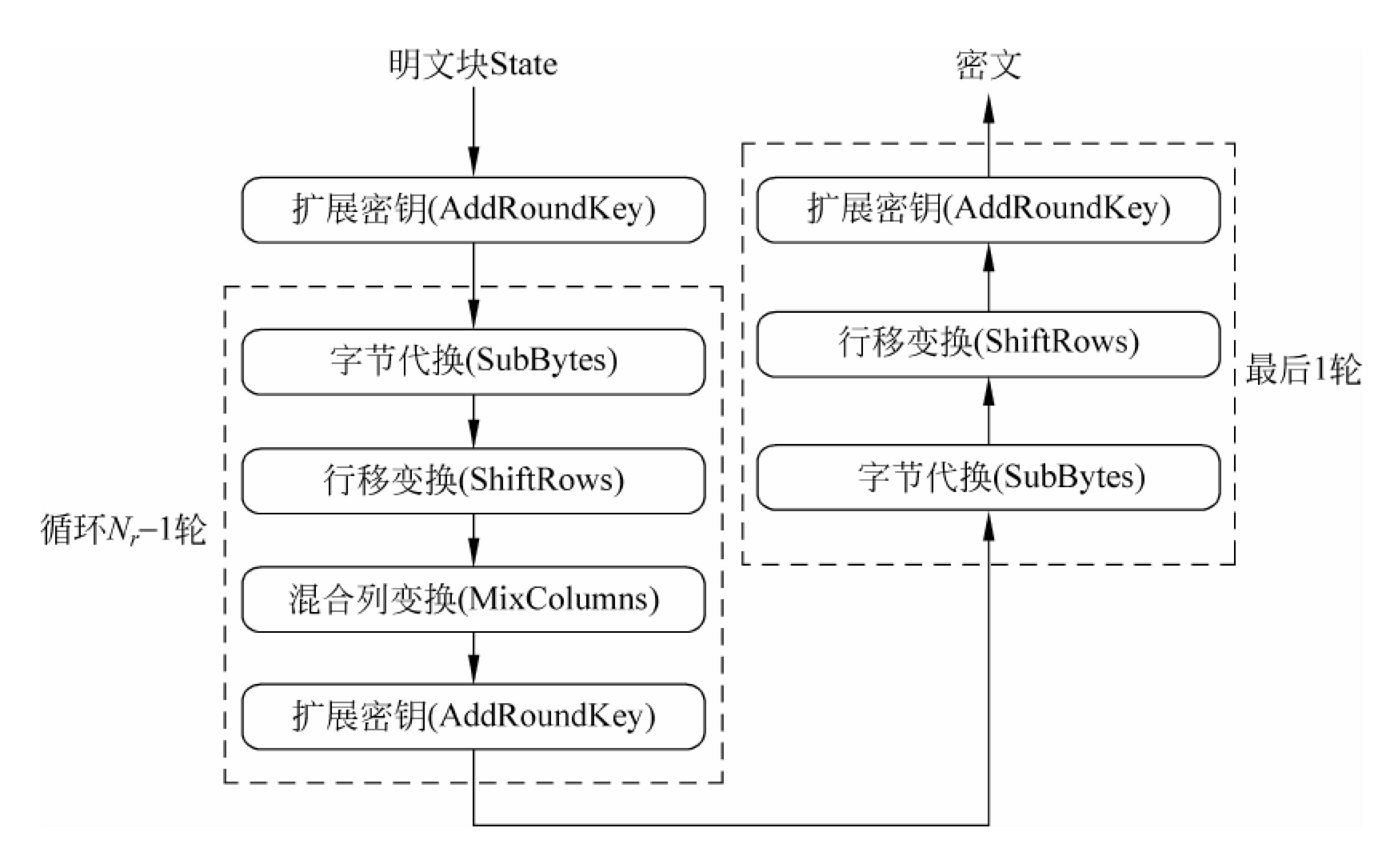


图 2.9 AES 算法的加密流程

1. 字节代换

字节代换是作用在字节上的一种非线性变换，它的实质就是置换。相当于 DES 中的 *S* 盒。通常实现 SubBytes 有两种方式：直接 *S* 盒置换和对 State 求模逆再进行仿射变换。

(1) *S* 盒置换：将 State 矩阵中的每个字节替换成一个由 *S* 盒决定的新字节。*S* 盒是一个固定的 16×16 矩阵，它的元素由 0x00～0xFF 组成，如表 2.16 所示。

例 2.8 假设 State[0,1]的值是 0x40，求 SubBytes(40)。

那么让 *S* 盒表中的 x 行等于左边的数字(4)并让 y 列等于右边的数字(0)，然后用 x 和 y 作为索引查找出 *S* 盒表中置换值，则 SubBytes(40) =0x09。

表 2.16 AES 加密的 *S* 盒

		y															
	hex	**0**	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	**63**	7c	77	7b	f2	6b	6f	c5	30	01	67	2d	fe	d7	ab	76
	1	**ca**	b2	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	**b7**	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	**04**	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	**09**	**b3**	**2c**	**1a**	**1b**	**6e**	**5a**	**a0**	**52**	**3b**	**d6**	**b3**	**29**	**e3**	**2f**	**84**
	5	**53**	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	**d0**	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
x	7	**51**	a3	40	af	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	**cd**	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	**60**	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	**e0**	32	3e	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	**e7**	c1	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	**ba**	78	25	2c	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	**70**	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	**e1**	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	**8c**	a1	89	0d	bf	bf	42	68	41	99	2d	0f	b0	54	bb	16

(2) 字节变换的第二种方式，相比第一种方式更复杂，但是在软、硬件实现时，消耗资源更少。下面针对第二种方式的介绍也体现了 S 盒的设计原理。

如果 $GF(2^8)$ 域上的二进制多项式 $a(x)$ 与 $b(x)$ 满足条件：

$$a(x)b(x) = 1 \bmod m(x)$$

那么 $a(x)$ 的模逆 $a(x)^{-1}=b(x)$。若 $b(x)=b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0$，则字节 a 的逆 a^{-1} 为 $b_7b_6b_5b_4b_3b_2b_1b_0$。

字节代换包括两个变换过程：首先对每个字节 a 在 $GF(2^8)$ 域运用欧几里德扩展定理分别求模逆，然后对所求得的模逆结果进行仿射变换。

仿射变换的运算规则如下：

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

其中，$b_7b_6b_5b_4b_3b_2b_1b_0$ 为字节 a 的逆 a^{-1}。

例 2.9　当前 State 的值是：

$$\begin{bmatrix} 19 & a0 & 9a & e9 \\ 3d & f4 & c6 & f8 \\ e3 & e2 & 8d & 48 \\ be & 2b & 2a & 08 \end{bmatrix}$$

求其 SubBytes 变换。

第一种方式：S 盒查表方式，将矩阵中每个元素按 S 盒查表得到结果如下。

$$\begin{bmatrix} d4 & e0 & b8 & 1e \\ 27 & bf & b4 & 41 \\ 11 & 98 & 5d & 52 \\ ae & f1 & e5 & 30 \end{bmatrix}$$

第二种方式：假设需要求解 SubBytes(19)(第 1 行第 9 列)，那么步骤如下。

首先，对 19(00011001)用欧几里德扩展定理求模逆。

在 $GF(2^8)$ 域中，“19”可表示成 (x^4+x^3+1)，且 $m(x)=x^8+x^4+x^3+x+1$ 或 $(11B)_{16}$，则：

$$\begin{aligned} 100011011 &= 11111 \cdot 11001 + 1100 \\ 11001 &= 10 \cdot 1100 + 1 \\ 1100 &= 1100 \cdot 1 \end{aligned}$$

所以：

$$y_1 = 0$$

$$y_2 = 1$$
$$y_3 = 0 - 1 \cdot 11111 = 11111$$
$$y_4 = 1 - 11111 \cdot 10 = 111111 = 0x3f$$

因此，19^{-1}为 0x3f。

其次，仿射变换过程。

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

即得 SubBytes (19)为 0xd4。

这种利用 GF(2^8)域上先模逆再求映射来构造 S 盒的好处：表述简单，使人相信 AES 算法没有设置“后门”，具有良好的抗差分和线性分析能力。仿射变换过程的目的是为了复杂化 S 盒的代数表达，以防止代数插值攻击，提高算法的安全性。

2. 行移变换

行移变换(ShiftRows)是指加密过程中，将对数据块的第 0～3 行进行循环左移运算。向左移动的位数与加密的数据块有关，行移变换的位参数如表 2.17 所示。

表 2.17 行移变换的位参数

N_b	移 位 值			
	C_0	C_1	C_2	C_3
4	0	1	2	3
6	0	1	2	3
8	0	1	3	4

例如，当 $N_b=4$ 时，则有：

$$\begin{bmatrix} 63 & 09 & cd & ba \\ ca & 53 & 60 & 70 \\ b7 & d0 & c0 & c1 \\ 04 & 51 & c7 & 80 \end{bmatrix} \to \begin{bmatrix} 63 & 09 & cd & ba \\ 53 & 60 & 70 & ca \\ c0 & c1 & b7 & d0 \\ 80 & 04 & 51 & c7 \end{bmatrix}$$

3. 混合列变换

经过以上变换后的矩阵元素都是 GF(2^8)域的成员，加法和乘法都是在 GF(2^8)域上进行。

混合列变换(MixColumns)是用多项式相乘定义的。这里的多项式的系数是用字节的十六进制表示。MixColumns 变换可以表示为：

$$\text{MixColumns}(d_{ij}) = C_{ij} \cdot S_{ij}$$

即：

$$\begin{bmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix}$$

其中，C_{ij} 作用到中间状态块 S_{ij} 的每一列，且 C_{ij} 是一个 4 字节元素的固定矩阵，矩阵中对应元素的乘法结果需要对 x^4+1 取模。

例 2.10　已知当前 State 是：

$$\begin{bmatrix} d4 & e0 & b8 & 1e \\ bf & b4 & 41 & 27 \\ 5d & 52 & 11 & 98 \\ 30 & ae & f1 & e5 \end{bmatrix}$$

求其 MixColumns 变换后的结果。

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} d4 & e0 & b8 & 1e \\ bf & b4 & 41 & 27 \\ 5d & 52 & 11 & 98 \\ 30 & ae & f1 & e5 \end{bmatrix}$$

由于：

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} d4 & e0 & b8 & 1e \\ bf & b4 & 41 & 27 \\ 5d & 52 & 11 & 98 \\ 30 & ae & f1 & e5 \end{bmatrix} = 04$$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} d4 & e0 & b8 & 1e \\ bf & b4 & 41 & 27 \\ 5d & 52 & 11 & 98 \\ 30 & ae & f1 & e5 \end{bmatrix} = 66$$

$$\vdots$$

经过矩阵相乘，最后得经 MixColumns 变换后的结果为：

$$\begin{bmatrix} 04 & e0 & 48 & 28 \\ 66 & cb & f8 & 06 \\ 81 & 19 & d3 & 26 \\ e5 & 9a & 7a & 4c \end{bmatrix}$$

4. 扩展密钥

扩展密钥(AddRoundKey)是将 N_b 个字节的轮密钥异或到相应状态 State 上。用于每一轮的 N_b 个字节的轮密钥取自扩展的密钥。

轮密钥通过扩展 AES 的密钥得到，接下来介绍密钥扩展的过程。

由 AES 算法加密的流程图可知，AES 需要 N_r+1 个轮密钥。因此需要把 N_k 个字节长的输入密钥 k 扩展成长度为 $N_k(N_r+1)$个字节的扩展后密钥 $w[\,]$，如图 2.10 所示。

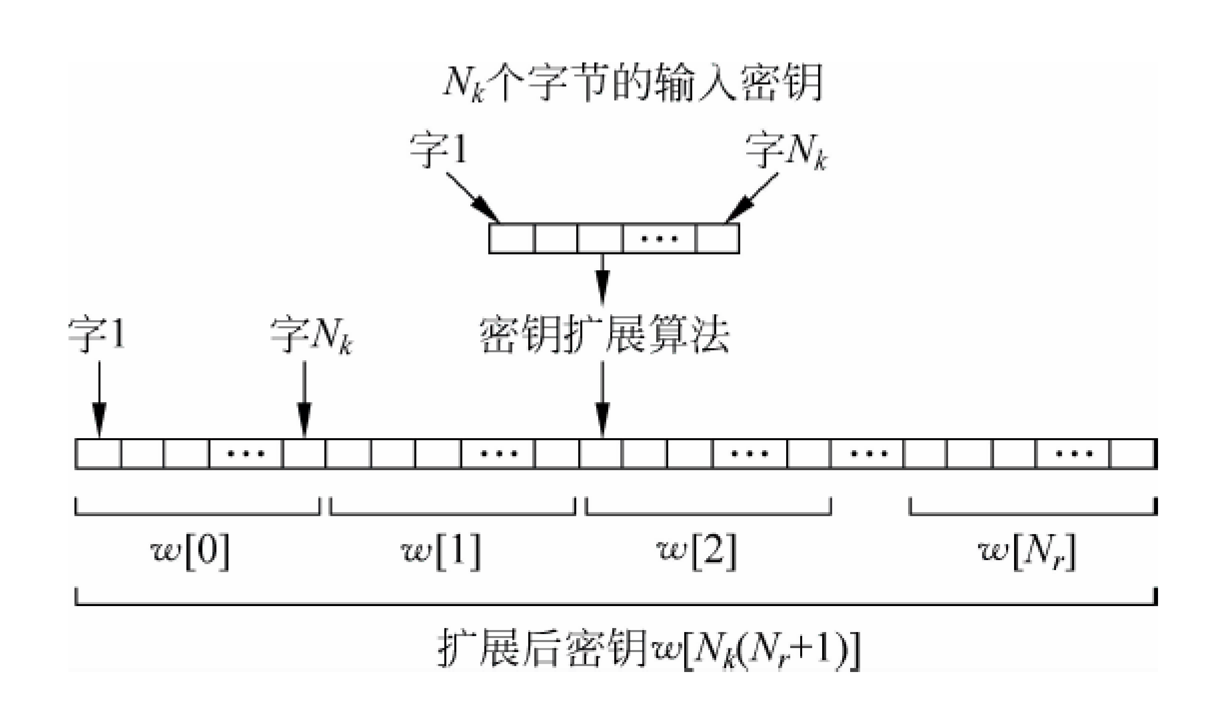


图 2.10　AES 密钥扩展

AES 扩展密钥涉及以下几个模块：

(1) RotWord()。它表示把一个 4 个字节的序列[a_0, a_1, a_2, a_3]左移一个字节变为[a_1, a_2, a_3, a_0]。

(2) SubWord()。对一个 4 字节的输入字[a_0, a_1, a_2, a_3]的每一个字节进行 S 盒变换，和前面讲的 S 盒变换是一样的。

(3) Rcon[]。轮常数 Rcon[i]表示表示 32 位字符串[x^{i-1}, 0x00, 0x00, 0x00]。这里 x=(02)，x 的 $i-1$ 次方是 x=(02)的 $i-1$ 次方的十六进制表示，于是得到：

Rcon[1] = [01,00,00,00]
Rcon[2] = [02,00,00,00]
Rcon[3] = [04,00,00,00]
Rcon[4] = [08,00,00,00]
Rcon[5] = [10,00,00,00]
Rcon[6] = [20,00,00,00]
Rcon[7] = [40,00,00,00]
Rcon[8] = [80,00,00,00]
Rcon[9] = [1b,00,00,00]
Rcon[10] = [36,00,00,00]

(4) 扩展密钥生成。根据 $N_k \leqslant 6$ 和 $N_k > 6$ 两种不同的情况，密钥扩展算法是不同的。

当 $N_k \leqslant 6$ 时，密钥扩展算法的伪代码如下：

```
for i: = 0 to Nk - 1
w[i] = (k0,i, k1,i, k2,i, k3,i)                  // kj,i是一个字节
for i: = Nk to NkNr - 1
temp = w[i - 1]
    if 0 ! = ( i mod Nk ) ,then w[i] = temp XOR w[i - Nk] ;
    if 0 = ( i mod Nk ) ,then w[i] = SubWord(RotWord(temp)) XOR w[i - Nk] XOR Rcon[i/Nk] ;
```

当 $N_k > 6$ 时，密钥扩展算法的伪代码如下：

```
for i: = 0 to Nk - 1
w[i] = (k0,i, k1,i, k2,i, k3,i)                  // kj,i是一个字节
for i: = Nk to NkNr - 1
temp = w[i - 1]
```

```
if0 ! = ( i mod Nk ) ,then w[i] = temp XOR w[i - Nk] ;
if 4 = ( i mod Nk ) ,then w[i] = SubWord(temp) XOR w[i - Nk] ;
```

这样就得到了长度为 $N_k(N_r+1)$个字的扩展后密钥 $w[N_k(N_r+1)]$。

接下来是子密钥的选择过程：加密时，第 i 个子密钥就是 $W_{N_b \times i} W_{N_b \times i+1} W_{N_b \times i+2} \cdots W_{N_b \times i+1-1}$；解密时的对应关系相反。

以轮数为 N_r、密钥 k 长度 128 位为例，AES 算法可完整描述如下：

(1) 子密钥生成。无论是 AES 加密还是解密，首先进行密钥扩展，即把 N_k 个字节长的输入密钥 k 扩展成长度为 $N_k(N_r+1)$个字节的扩展后密钥 w。结合前面关于 AddRoundKey 所讲，由于本例中 $N_k=4$，所以应按第一种情况($N_k \leqslant 6$)进行密钥扩展。过程如下。

将 4 字节密钥 k 排列成 4×4 字节的矩阵，即按以下方法扩展成 44 列：分别记扩展后密钥 w 的前 4 列为 $k(0)$、$k(1)$、$k(2)$及 $k(3)$。w 的后 40 列按 i 能否被 4 整除，相应地分两种情况进行扩展。

当 $i \geqslant 4$ 时，若 $i \bmod 4 \neq 0$，则：

$$k(i) = k(i-1) \text{ XOR } k(i-4)$$

若 $i \bmod 4 = 0$，则：

$$k(i) = \text{SubWord}(\text{RotWord}(k(i-1))) \text{ XOR } k(i-4) \text{ XOR } \text{Rcon}((i-4)/4)$$

此时，密钥扩展过程用到上面的 RotWord()、SubWord()和 Rcon()3 种变换。

经过以上步骤之后，那么第 i 轮的轮密钥则为：

$$k(4i), k(4i+1), k(4i+2), k(4i+3) \quad 0 \leqslant i \leqslant N_r$$

(2) 加密过程。对明文块 State 循环执行以下操作：

① 扩展密钥(AddRoundKey)；

② 依次迭代进行(N_r-1)轮 SubBytes、ShiftRows、MixColumns、AddRoundKey；

③ 在最后一轮中进行 SubBytes、ShiftRows、AddRoundKey。

(3) 解密过程。由于在 GF(2^8)域进行 AES 加密的各变换均可逆，因此解密算法的结构和加密的结构相同，相应的变换为加密时的逆变换。

例 2.11　加密过程 MixColumns 选择的固定矩阵：

$$C_{ij} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

其逆为：

$$\begin{bmatrix} 0e & 09 & 0d & 0b \\ 0b & 0e & 09 & 0d \\ 0d & 0b & 0e & 09 \\ 09 & 0d & 0b & 0e \end{bmatrix}$$

综合 AES 加密过程，不难得出以下两点：

① InvSubByte 作用于每个字节；InvShiftRows 不改变字节的值，仅进行移位操作；InvSubByte 和 InvShiftRows 的顺序是可以交换的。

② 逆混合列变换满足：InvMixColumns(State ⊕ Round_Key) = InvMixColumns

(State)⊕InvMixColumns(Round_Key)。

因此,除第一轮和最后一轮外,在循环过程的 N_r-1 轮中可以交换 AddRoundKey 和 InvMixColumns 的操作顺序。交换顺序之后的过程构成 AES 的解密算法。

AES 的解密流程如图 2.11 所示。

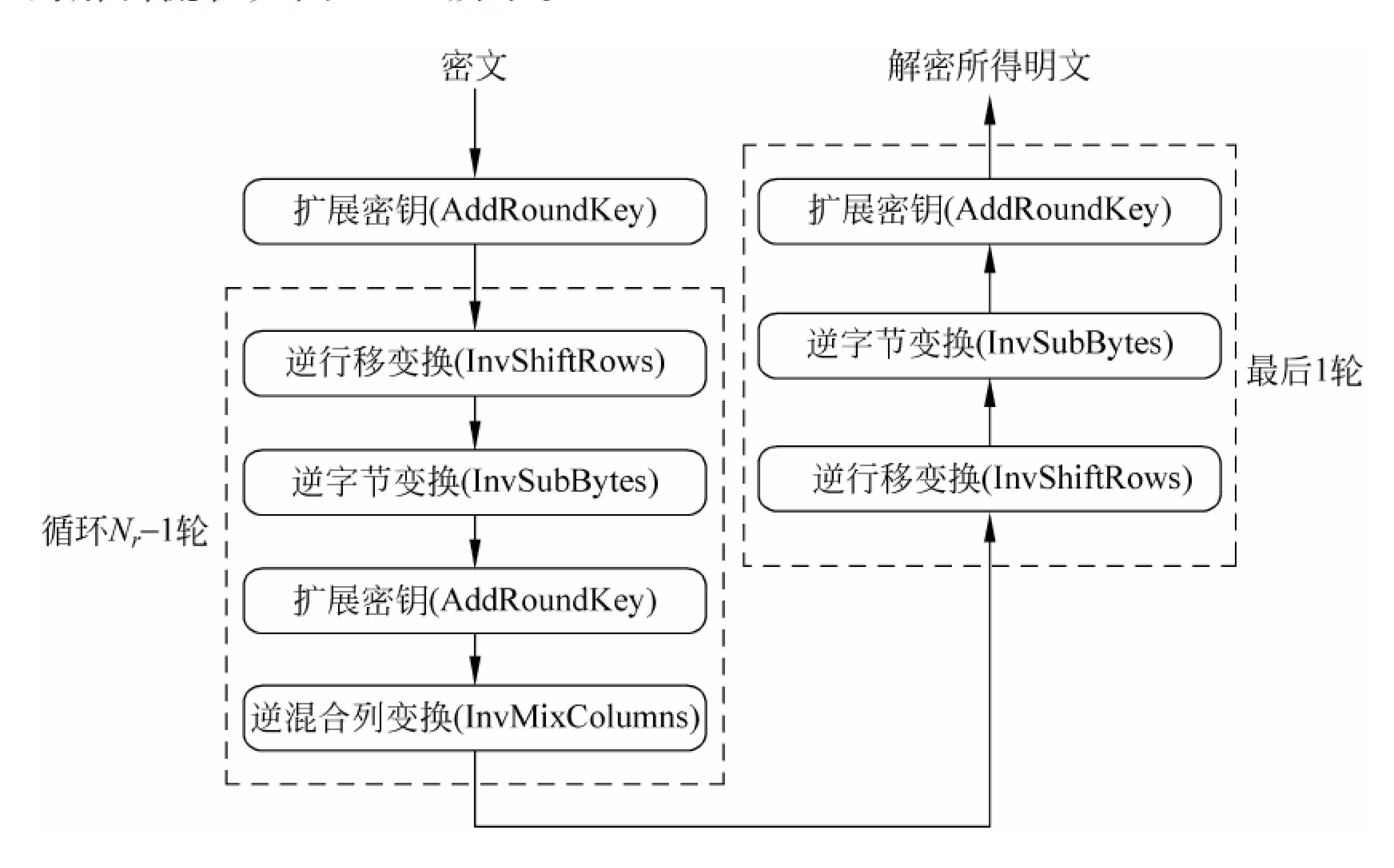


图 2.11　AES 算法的解密流程

2.4　加 密 模 式

分组密码加密的消息长度都是固定的,如 DES 为 64 位,AES 为 128 位、192 位或 256 位;而在实际应用中要处理的消息往往又是任意长度的,于是不可避免地引出一个重要的实施问题,即加密模式问题。

1980 年,NIST 公布了 4 种 DES 的工作模式:电码本(ECB)模式,密码分组链接(CBC)模式,密码反馈(CFB)模式和输出反馈(OFB)模式。其中,ECB 和 CBC 属于块加密,CFB 和 OFB 属于流加密。

下面以 DES 为例,对以上几种加密模式进行说明。

为便于描述,此处假设 P_i 是明文块(64 位),C_i 是密文块(64 位),K 是加密密钥,$E_K(P_i)$ 是加密函数,$D_K(C_i)$是解密函数。

1. ECB 模式

ECB(Electronic Code Book)模式是最常用的加密模式之一,它对明文分组不进行任何处理,直接进行加密,得到的结果就是密文。

ECB 模式加密算法表示为:

$$C_i = E_K(P_i)$$

ECB 模式解密算法表示为:

$$P_i = D_K(C_i)$$

ECB 模式加密过程如图 2.12 所示。

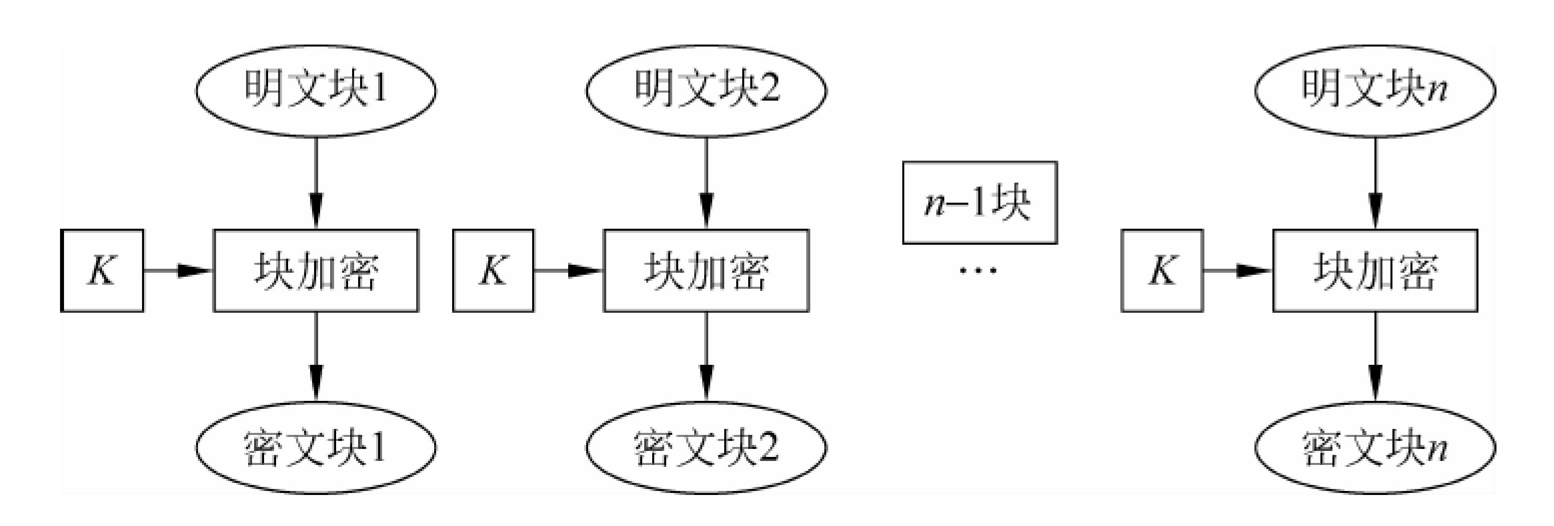


图 2.12　ECB 加密模式图解

ECB 模式解密过程如图 2.13 所示。

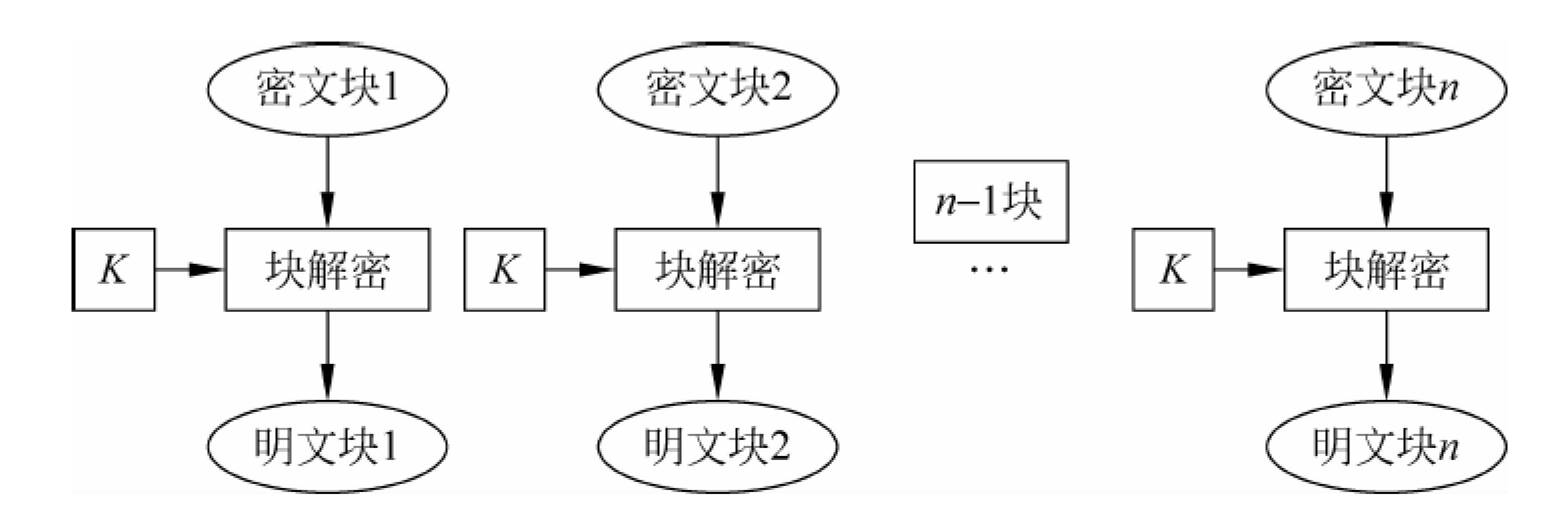


图 2.13　ECB 解密模式图解

例 2.12　明文是"securitysecurity"，密钥是"12345678"，如果采用 ECB 模式进行加密生成的密文是：

"5bd92773264cf76b5bd92773264cf76b"

不难看出，加密后的密文"5bd92773264cf76b5bd92773264cf76b"中出现了重复的部分，说明 ECB 的加密是分组独立进行的，不能隐蔽数据模式。

ECB 模式有以下优点：

(1) 每个分组可以独立进行加密，不必按次序进行；

(2) 可并行运算，速度快，易于标准化；

(3) 一个错误仅仅会对一个密文块产生影响。

同时，ECB 模式又有以下缺点：

(1) 相同的明文分组永远被加密成相同的密文分组；

(2) 不能隐蔽数据模式。

2. CBC 模式

1976 年，IBM 率先提出密码分组链接(Cipher-Block Chaining，CBC)模式。与 ECB 模式不同的是：CBC 模式引入随机初始向量，并将当前的明文分组和前一个密文分组异或后作为新的明文分组进行加密，这样使得相同的明文分组可以产生不同的密文分组。设 IV 是初始向量，第一个块的下标为 1。

CBC 模式加密算法可表示为：

$$C_0 = \mathrm{IV}$$
$$C_i = E_K(P_i \oplus C_{i-1}) \quad 1 \leqslant i \leqslant n$$

其中，$C_0 = \mathrm{IV}$ 是随机初始变量。

CBC 模式解密算法可表示为：

$$C_0 = \mathrm{IV}$$
$$P_i = D_K(C_i) \oplus C_{i-1} \quad 1 \leqslant i \leqslant n$$

CBC 模式加密过程如图 2.14 所示。

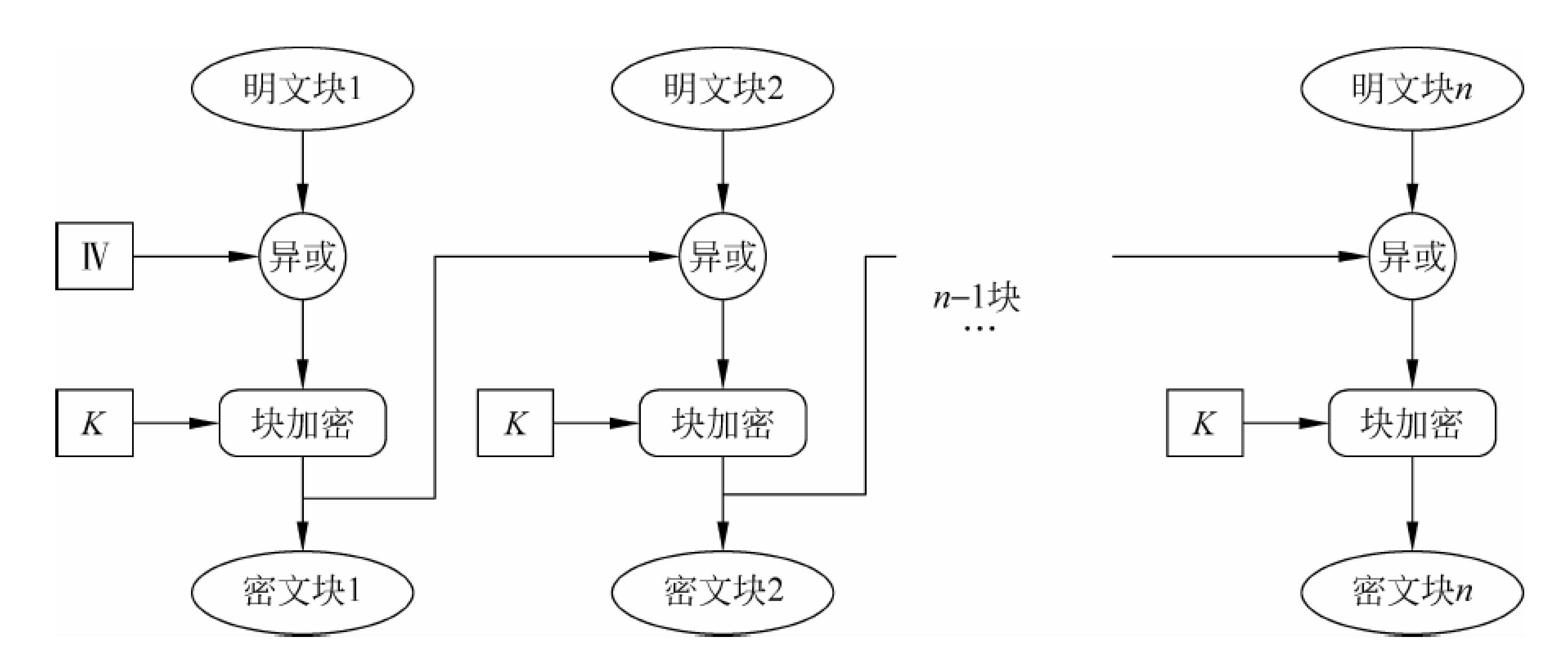


图 2.14 CBC 模式加密图解

CBC 模式解密过程如图 2.15 所示。

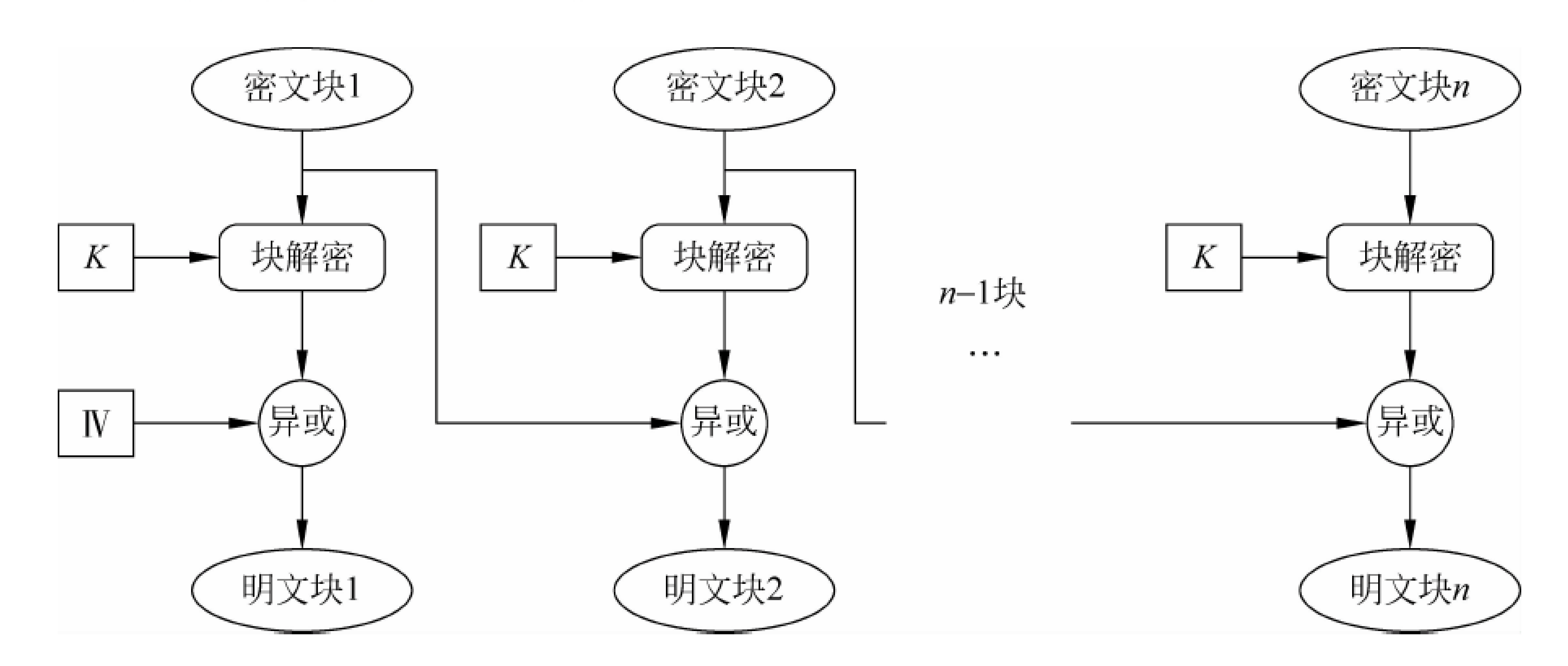


图 2.15 CBC 模式解密图解

由于 CBC 模式引入随机初始向量，隐藏了明文的数据模式，在一定程度上能防止数据篡改，安全性好于 ECB；不过 CBC 模式不太适合并行计算，有可能导致错误传播。

例 2.13 设明文是"securitysecurity"，密钥是"12345678"，随机初始变量 IV 是"12345678"，按 CBC 模式对明文进行加密的过程如下：

将明文"security"转换成十六进制的 ASCII 码："7365637572697479"。

将密钥"12345678"转换成十六进制的 ASCII 码："3132333435363738"。

首先用密钥对明文"security"进行加密得到密文：

"1b57ee503eb96d9c"

将密文"1b57ee503eb96d9c"与"7365637572697479"(IV："12345678")异或得到：

"68328d254cd019e5"

然后用密钥"3132333435363738"对明文"68328d254cd019e5"进行加密得到密文：

"dc0bee199cd46d20"

于是，采用 CBC 模式对明文“securitysecurity”的加密最终结果是：

“1b57ee503eb96d9cdc0bee199cd46d20”

3. CFB 模式

CFB(Cipher Feed Back)模式类似于 CBC 模式，但是它采用了密钥流生成器，将分组密码转化成流密码。在 CFB 模式中，先加密前一个块的密文，然后将得到的结果与明文块异或产生当前分组，所得结果反馈进入移位寄存器作为下一阶段的输入。

CFB 模式的自同步能力和 CBC 模式是相同的；若密文的一整块发生错误，CFB 模式仍能解密大部分数据，而仅有一位数据出错。

CFB 模式加密算法可表示为：

$$C_i = E_K(C_{i-1}) \oplus P_i$$

其中，$C_0 = \mathrm{IV}$ 是随机初始变量。

CFB 模式解密算法可表示为：

$$P_i = E_K(C_{i-1}) \oplus C_i$$

CFB 模式加密过程如图 2.16 所示。

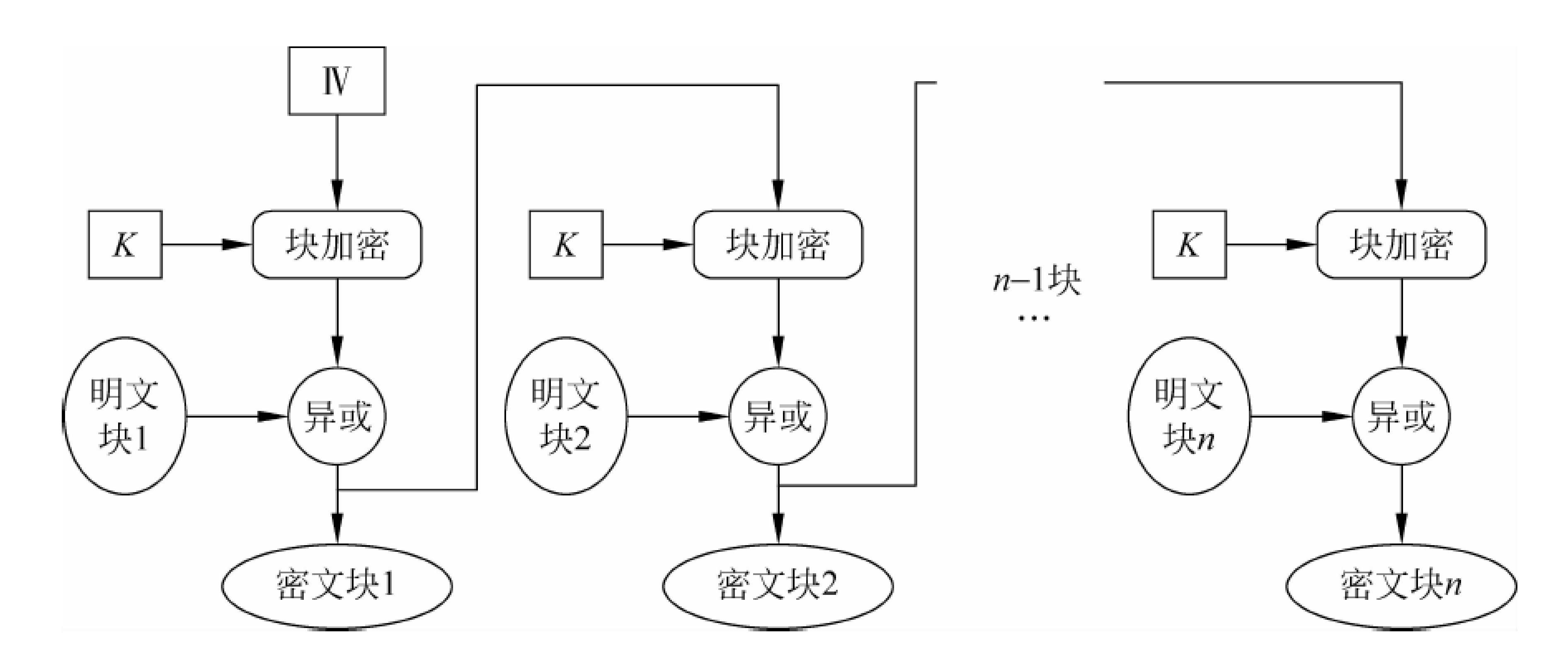


图 2.16　CFB 模式加密图解

CFB 模式解密过程如图 2.17 所示。

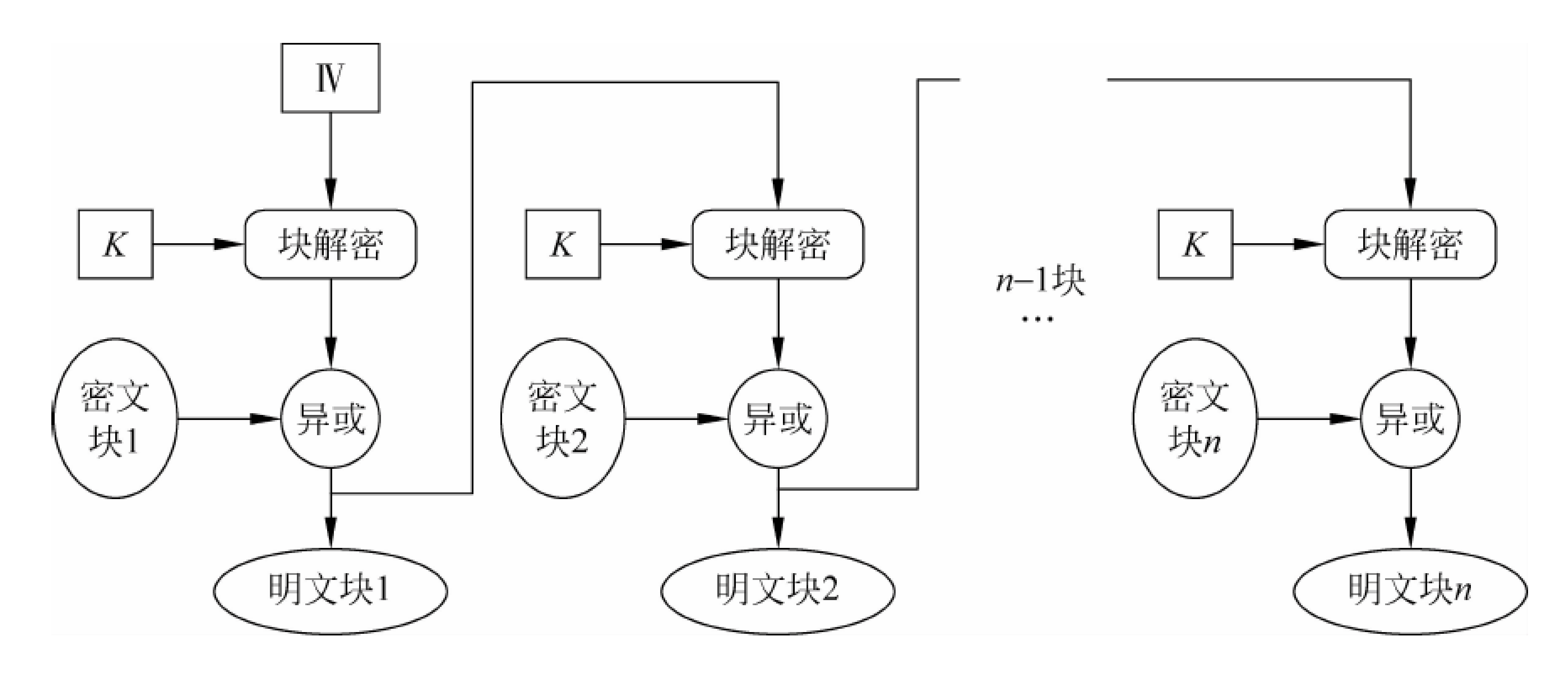


图 2.17　CFB 模式解密图解

CFB 模式具有自同步的能力，可以加密任意长度的消息，非常适合数据流的加密。不过，与 CBC 模式一样，CFB 模式也不太适合并行计算，并且存在错误传播。

例 2.14 明文是"securitysecurity",密钥是"12345678",随机初始变量 IV 是"12345678",按 CFB 模式对明文进行加密的过程如下:

将明文"security"转化成十六进制的 ASCII 码:"7365637572697479"。

将密钥"12345678"转化成十六进制的 ASCII 码:"3132333435363738"。

"3132333435363738"(IV"12345678")被密钥"3132333435363738"加密后密文是:

"96d0028878d58c89"

将"96d0028878d58c89"与明文"7365637572697479"异或得到:

"e5b561fd0abcf8f0"

再用密钥"3132333435363738"对"e5b561fd0abcf8f0"进行加密得到的密文:

"ff03803dd2c97ca1"

"ff03803dd2c97ca1"与明文的十六进制字符串"7365637572697479"异或后得到:

"8c66e348a0a008d8"

于是,最终的密文是:

"e5b561fd0abcf8f08c66e348a0a008d8"

4. OFB 模式

OFB(Out Feed Back)模式的加密模式与 CFB 的模式相似,不同之处是 OFB 将加密算法的输出反馈到了移位寄存器,而 CFB 模式将密文单元反馈到了移位寄存器。

OFB 模式加密算法可以表示为:

$$Z_i = E_K(Z_{i-1}), \quad C_i = Z_i \oplus P_i$$

其中,$Z_0 = \text{IV}$ 是随机初始变量。

OFB 模式解密算法可以表示为:

$$Z_i = E_K(Z_{i-1}), \quad P_i = Z_i \oplus C_i$$

OFB 模式加密过程如图 2.18 所示。

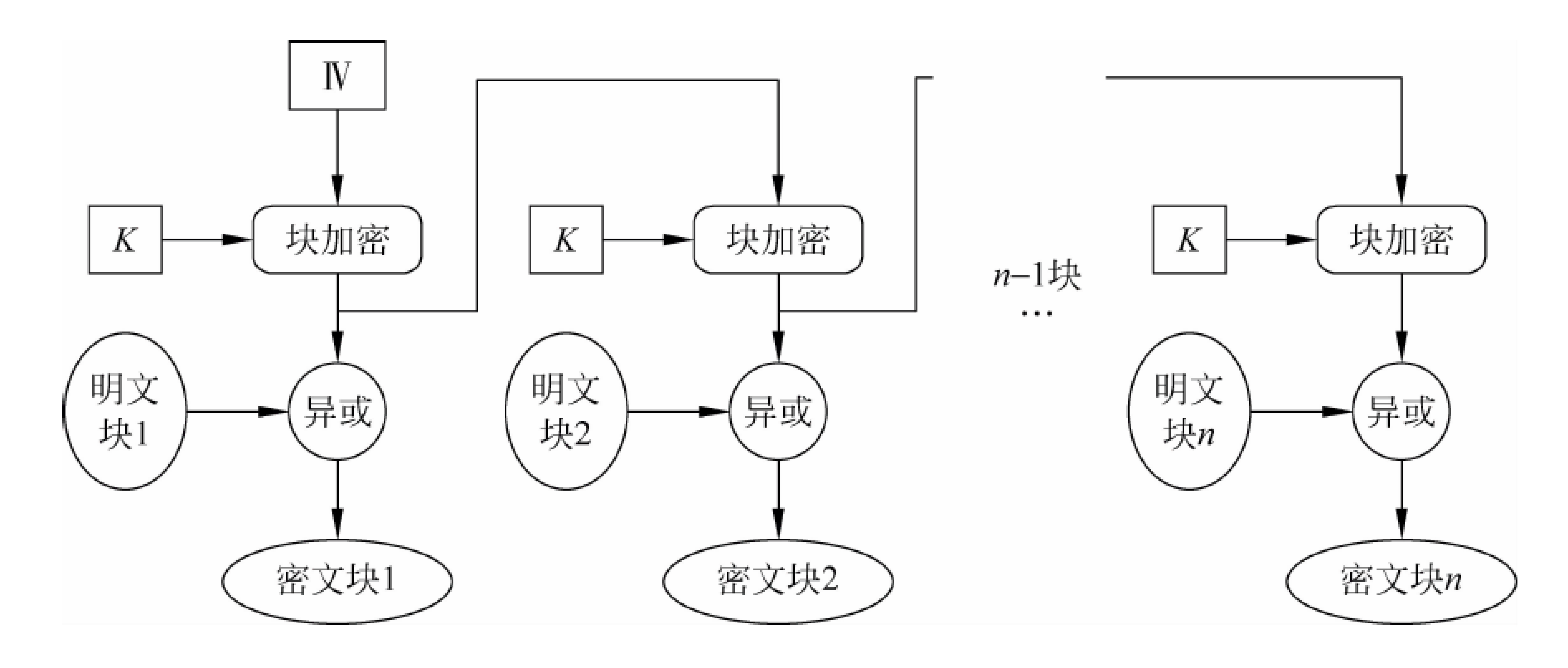


图 2.18 OFB 模式加密图解

OFB 模式解密过程如图 2.19 所示。

OFB 模式的优点是错误传播小;缺点是对通信双方的同步要求高,也不适合并行计算。

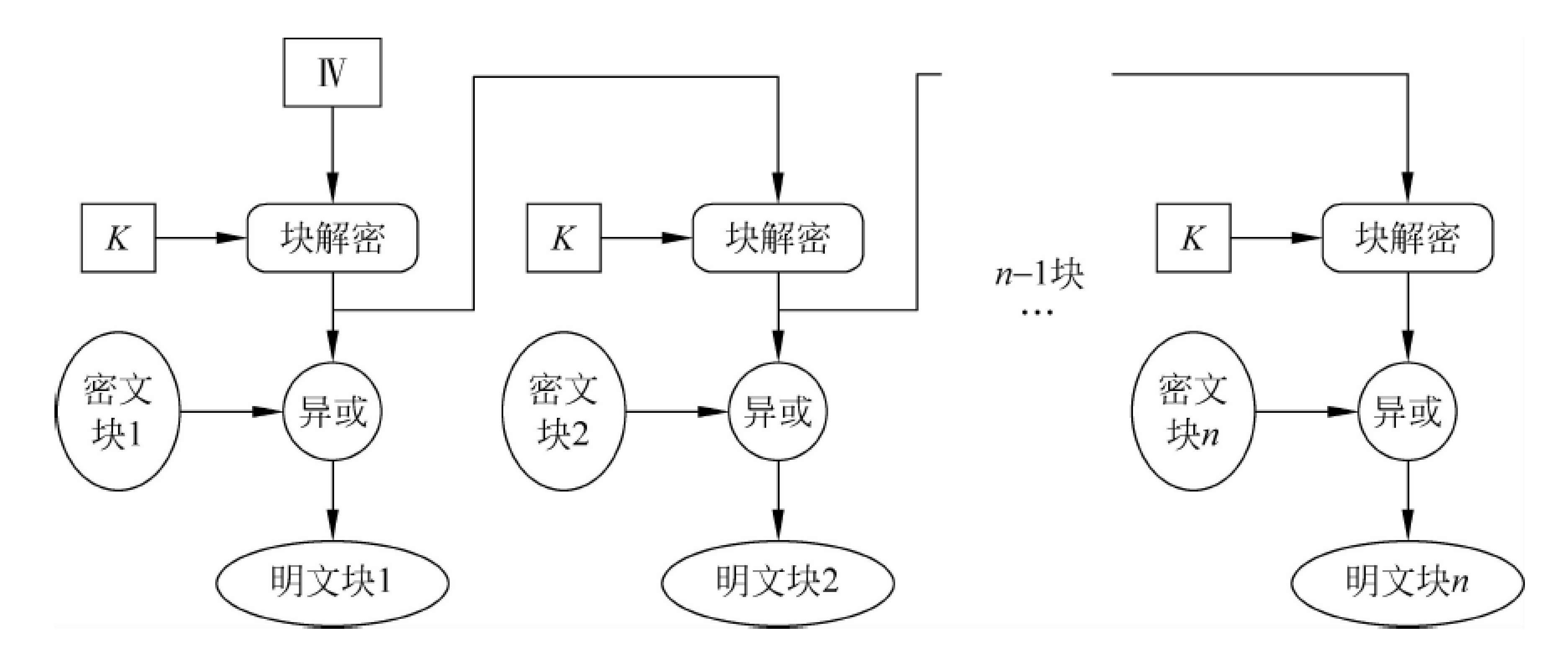


图 2.19　OFB 模式解密图解

例 2.15　设明文是“securitysecurity”，密钥是“12345678”，随机初始变量 IV 是“12345678”，按 OFB 模式对明文进行加密的步骤如下：

将明文“security”转化成十六进制的 ASCII 码：“7365637572697479”。

将密钥“12345678”转化成十六进制的 ASCII 码：“3132333435363738”。

“3132333435363738”(IV“12345678”)被密钥“3132333435363738”加密后的密文是：

“96d0028878d58c89”

将“96d0028878d58c89”与明文“7365637572697479”异或得到：

“e5b561fd0abcf8f0”

再用密钥“3132333435363738”对“96d0028878d58c89”进行加密后得到的密文：

“e97598df296f12aa”

“e97598df296f12aa”与明文“7365637572697479”异或后得到：

“9a10fbaa5b0666d3”

所以，加密后的密文是：

“e5b561fd0aa1e5f0 9a10fbaa5b0666d3”

2.5　习　　题

1. 使用 DES 算法对明文 84040d5158059e25 进行加密，密钥为 19a3b3fd5ee69dca。
2. 使用 DES 算法对密文 6b968d81663f71bf 进行解密，密钥为 e33479fea9a76ca7。
3. 找出所有二进制系数的 7 次本原多项式。
4. 使用 DES 算法对明文 84040d5158059e2584040d5158059e2584040d5158059e25 进行加密，密钥为 19a3b3fd5ee69dca，加密模式采用 ECB 模式。
5. 使用 DES 算法对明文 84040d5158059e2584040d5158059e2584040d5158059e25 进行加密，密钥为 19a3b3fd5ee69dca，加密模式采用 CBC 模式。
6. 使用 10 回合的 AES 算法对明文 10abababababababababababababab10 进行加密，密钥是 11111111111111111111111111111111。

7. 使用10回合的AES算法对密文62abdcd5431a2d2f1002ac23ca59902f进行解密，密钥是22222222222222222222222222222222。

8. 使用10回合的AES算法对明文10ababababababababababababab10进行加密，密钥是11111111111111111111111111111111，加密模式采用ECB模式。

9. 使用10回合的AES算法对明文10ababababababababababababab10进行加密，密钥是11111111111111111111111111111111，加密模式采用CBC模式。

第 3 章　基于因式分解的公钥密码系统

1976 年，Diffie 和 Hellman 在“密码学的新方向”一文中，率先提出公钥密码体制的新思想。在公钥密码体制中，加密和解密由一对密钥来完成，分别称为公钥和私钥，公钥可以公开，私钥则需保密。

公钥密码系统也被称为非对称密码系统。

1978 年，Rivest、Shamir 和 Adleman 在论文 *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems* 中首次提出一种比较完善的公钥密码体制，后来被称为 RSA 公钥密码体制。

RSA 公钥密码体制的安全性依赖于大数的因式分解难题。

3.1　欧拉函数与循环群

在介绍 RSA 公钥密码体制前，先来介绍相关的数学基础知识。

欧拉函数： 自变量为正整数 n 的函数 $\varphi(n)$ 称为欧拉函数，$\varphi(n)$ 表示小于 n 且与 n 互素的正整数的个数。

显然，当 n 为素数时，$\varphi(n)=n-1$。

下面的定理有助于求解欧拉函数的值。

定理 3.1　若 $\gcd(a,b)=1$，则 $\varphi(ab)=\varphi(a)\varphi(b)$。

证明： 任意 $x(0\leqslant x<ab)$ 与数对 (k_1,k_2) 一一对应：

$$\begin{cases} x \equiv k_1 \pmod a \\ x \equiv k_2 \pmod b \end{cases}$$

因为 $<0>_m$、$<1>_m$、…、$<ab-1>_m$ 一共是 ab 个数，(k_1,k_2) 也一共有 ab 种组合，且都各不相同。

对于 $0\leqslant x<ab$ 的任意 x，x 与 ab 互素等价于 x 与 a 和 b 都互素，用反证法易证。

而 x 与 a 和 b 都互素又等价于 k_1 与 a 互素且 k_2 与 b 互素。

因此，与 ab 互素的数 x 的个数等于数对 $\{(k_1,k_2)\mid \gcd(k_1,a)=1,\gcd(k_2,b)=1\}$ 的个数，即：

$$\varphi(ab)=\varphi(a)\varphi(b)$$

命题得证。

定理 3.2　如果 $n=p_1^{a_1}p_2^{a_2}\cdots p_m^{a_m}$，其中 p_i 为素数，$0<i\leqslant m$，那么：

$$\varphi(n)=n\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_m}\right)$$

证明： 首先证明 $\varphi(p^a)=p^a\left(1-\frac{1}{p}\right)$。

任意 $x(0\leqslant x<n)$ 要么是 p 的倍数，要么与 p^a 互素。其中，p 的倍数分别是 0、p、$2p$、…、$(p^{a-1}-1)p$，一共有 p^{a-1} 个，因此与 p^a 互素的数是 $p^a-p^{a-1}=p^a\left(1-\frac{1}{p}\right)$ 个。

根据前面的定理可得：

$$\begin{aligned}\varphi(n) &=\varphi(p_1^{a_1}p_2^{a_2}\cdots p_m^{a_m})\\ &=\varphi(p_1^{a_1})\varphi(p_2^{a_2})\cdots\varphi(p_m^{a_m})\\ &=p_1^{a_1}\left(1-\frac{1}{p_1}\right)\cdot p_2^{a_2}\left(1-\frac{1}{p_2}\right)\cdot\cdots\cdot p_m^{a_m}\left(1-\frac{1}{p_m}\right)\\ &=p_1^{a_1}p_2^{a_2}\cdots p_m^{a_m}\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_m}\right)\\ &=n\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_m}\right)\end{aligned}$$

命题得证。

定理 3.3 （欧拉定理）已知 a、n 为整数，如果 $\gcd(a,n)=1$，则有：

$$a^{\varphi(n)}=1(\text{mod } n)$$

证明：设 $\varphi(n)=k$，假设 r_1、r_2、…、r_k 是与 n 互素的模 n 的同余类的主余数。

由于 $\gcd(a,n)=1$，ar_1、ar_2、…、ar_k 也应该是互素的，而且对于模 n 两两不同余，因此

$$(ar_1)(ar_2)\cdots(ar_k)=a^k r_1 r_2\cdots r_k=r_1 r_2\cdots r_k(\text{mod } n)$$

$$(a^k-1)r_1 r_2\cdots r_n\equiv 0(\text{mod } n)$$

因为 r_1、r_2、…、r_k 与 n 互素，所以 $a^k\equiv 1(\text{mod } n)$，命题得证。

例 3.1 设 $n=16$，$a=9$，满足 $\gcd(a,n)=1$，与 n 互素的数分别为 1、3、5、7、9、11、13、15，所以 $\varphi(n)=8$，于是：

$$a^{\varphi(n)}=9^8=43\,046\,721=1(\text{mod } 16)$$

定理 3.4 （费马(Fermat)定理）设 a 为整数，n 是素数，如果 $\gcd(a,n)=1$，则有：

$$a^{n-1}=1(\text{mod } n)$$

证明：因为 n 是素数，所以欧拉函数 $\varphi(n)=n-1$，再根据欧拉定理即可证明。

例 3.2 已知 $n=7$，那么下列等式成立：

$$2^6=64=1(\text{mod } 7)$$

$$3^6=729=1(\text{mod } 7)$$

$$4^6=4096=1(\text{mod } 7)$$

$$5^6=15\,625=1(\text{mod } 7)$$

$$6^6=46\,656=1(\text{mod } 7)$$

群、半群、含么半群、交换群：G 为非空集合，如果在 G 上定义的二元运算“$*$”满足：

(1)（封闭性）对于任意 $a,b\in G$，$a*b\in G$；

(2)（结合律）对于任意 $a,b,c\in G$，有 $(a*b)*c=a*(b*c)$；

(3)（么元）存在么元 e 使得对于任意 $a\in G$，$e*a=a*e=a$；

(4)（逆元）对于任意 $a\in G$ 存在逆元 a^{-1}，使得 $a^{-1}*a=a*a^{-1}=e$；

则称 $(G,*)$ 是群(Group)，有时简称 G 是群；如果仅满足(1)和(2)则称 G 是一个半群(Semigroup)；如果仅满足(1)、(2)和(3)则称 G 是一个含么半群(Monoid)。

(5) 如果群 $(G,*)$ 还满足交换律：对于任意 $a,b\in G$，有 $a*b=b*a$；
则称 G 为交换群或阿贝尔群(Abel Group)。

例 3.3　Z 是整数集合，$+$ 和 $*$ 分别是整数的加法和乘法运算，$(Z,+)$ 是群，且 $(Z,*)$ 不是群，这是因为除 1 和 -1 外的其他元素都没有逆元。

例 3.4　设 $G=\{1,-1\}$，$*$ 是数的乘法，则 $(G,*)$ 是群。

Klein 四元群：设集合 $K=\{e,a,b,c\}$，K 中的二元运算"$*$"由表 3.1 中的 Klein 乘法表给出。

表 3.1　Klein 乘法表

$*$	e	a	b	c
e	e	a	b	c
a	a	e	c	b
b	b	c	e	a
c	c	b	a	e

则 $(K,*)$ 是群。

首先，不难验证 $(K,*)$ 满足结合律，幺元是 e，每个元素的逆元是它本身，所以 $(K,*)$ 是群，而且还是交换群。

例如，(幺群)$(\{0\},+)$ 是群。由于只包含幺元，所以该群又称幺群。

除了 $(\{0\},+)$ 是幺群外，$(\{1\},*)$ 也是幺群。

关于群的幺元和逆元有以下结论：

(1) 幺元是唯一的：设 G 有两个幺元分别为 e_1 和 e_2，则：

$$e_1 = e_1 * e_2 = e_2$$

(2) 逆元是唯一的：设 G 中元素 a 有两个逆元分别为 a_1^{-1} 和 a_2^{-1}，则：

$$a_1^{-1} = a_1^{-1} * e = a_1^{-1} * (a * a_2^{-1}) = (a_1^{-1} * a) * a_2^{-1} = e * a_2^{-1} = a_2^{-1}$$

(3) $a_1 * a_2$ 的逆元是 $a_2^{-1} * a_1^{-1}$。

(4) $a_1 * a_2 * \cdots * a_n$ 的逆元是 $a_n^{-1}a_{n-1}^{-1}\cdots a_2^{-1}a_1^{-1}$。

无限群、有限群、群的阶：设 $(G,*)$ 是群，G 的元素个数是无限时，G 称为无限群；G 的元素个数是有限时，G 称为有限群；G 的元素个数称为群 G 的阶。

元素的阶：满足 $a^n=e$ 的最小正整数 n，称为元素 a 的阶，记作 $o(a)$。如果 $a^n=e$ 永不成立，则称 a 的阶为无穷大。

定理 3.5　设 G 是群，$a\in G$，则 $a^m=1\Leftrightarrow o(a)\mid m$。

证明：如果 $a^m=1$，设 $o(a)=n\leqslant m$，$m=kn+r$，$0\leqslant r<n$，于是：

$$a^m = a^{kn+r} = a^{kn}a^r = a^r = 1$$

但是 a 的阶是 n 而不是 r，所以 $r=0$，$m=kn$，因此 $o(a)\mid m$；

反之，如果 $o(a)\mid m$，则 $m=kn$，于是：

$$a^m = a^{kn} = (a^n)^k = 1^k = 1$$

命题得证。

子群：设 S 是群 G 的一个非空子集，如果 S 对 G 的运算也构成群，则称 S 是 G 的子群(Subgroup)，记作 $S\leqslant G$。

循环子群、生成元：设 G 是群，$a \in G$，令 $H=\{a^k \mid k \in Z\}$，H 对于二元运算同时满足：

(1) 封闭性：对于任意的 a^{k_1}、$a^{k_2} \in G$，$a^{k_1} * a^{k_2} = a^{k_1+k_2} \in G$；

(2) 结合律：对于任意的 a^{k_1}、a^{k_2}、$a^{k_3} \in G$，$(a^{k_1} * a^{k_2}) * a^{k_3} = a^{k_1} * (a^{k_2} * a^{k_3})$；

(3) 幺元：幺元是 G 的幺元 e；

(4) 逆元：对于任意的 $a^k \in G$，逆元为 $a^{-k} \in G$，$a^k * a^{-k} = a^{-k} * a^k = e$。

因此 H 是 G 的子群，称为循环子群(Cyclic Subgroup)，记作 $<a>$，a 是该子群的生成元。

特别地，当 $G=<a>$ 时，称 G 是循环群，下面给出循环群的完整定义。

循环群：一个群 $(G, *)$ 称为循环群，如果存在一个元素 $\alpha \in G$，使得：

$$G = \{\alpha^n \mid n \in Z\}$$

例 3.5 $(Z, +)$ 是由 1 生成的循环群，因为任何整数都可以写成若干个 1 相加的形式。

实际上，循环群 $(Z, +)$ 的生成元只能是 1 和 -1，因为设生成元为 a，由于 $1 \in Z$，必存在 k 使得 $ka=1$，于是 $a=\pm 1$。

例 3.6 $(Z_p, +)$ 是循环群。

令 $p=6$，$Z_6=\{[0]_6, [1]_6, \cdots, [5]_6\}$，循环群 Z_6 的幺元是 $[0]$，元素 $[1]$ 是生成元，元素 $[1]$ 的阶是 6，元素 $[2]$ 的阶是 3，元素 $[3]$ 的阶是 2。

例 3.7 如果 p 是素数，那么 $(Z_p^*, *)$ 是循环群，其中 $Z^* = Z \setminus [0]_m$。

表 3.2 给出了当生成元为 2，$p=11$ 时 $2^b \bmod 11$ 的计算结果。可以看到 $2^{b+10} \bmod 11$ 与 $2^b \bmod 11$ 值相同，出现循环。

表 3.2 生成元为 2，$p=11$ 时的循环群

b	1	2	3	4	5	6	7	8	9	10	11
$2^b \bmod 11$	2	4	8	5	10	9	7	3	6	1	2

关于循环群 $(Z_p, +)$ 的生成元还有以下性质。

定理 3.6 $(Z_p, +)$ 的生成元只能是 $[a]_p$，其中 $\gcd(a, n)=1$。

证明：设 $[a]_p$ 是生成元，因为 $[a]_p$ 必有逆，即：

$$k[a]_p = 1$$

则存在 t 使得：

$$k[a]_p + tn = 1$$

说明 $\gcd(a, n)=1$，证毕。

3.2 RSA 原理

目前 RSA 已经发展成应用最广泛的公钥密码算法之一。RFC 小组在 RFC2313 中对 RSA1.5 版进行了详尽的描述之后，又在 RFC3447 中将 RSA 发展到了 2.1 版。

除 RFC3447 外，RSA 算法的应用标准或草案还包括以下内容。

(1) RFC2537：RSA/MD5 KEYs and SIGs in the Domain Name System (DNS)。

(2) RFC2792：DSA and RSA Key and Signature Encoding for the KeyNote Trust

Management System。

(3) RFC3110：RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS)。

RSA 公钥密码系统是建立在欧拉定理和大数因式分解难题的基础上的。下面是 RSA 公钥密码系统的具体描述。

RSA 公钥密码系统：设 $n=pq$，p 和 q 是素数。

明文空间和密文空间：

$$P = C = Z_n$$

密钥：

$$K = \{(n,p,q,a,b):ab \equiv 1(\bmod \varphi(n))\}$$

其中，n、b 是公钥；p、q、a 是私钥。

RSA 加密算法：

$$e_K(x) = x^b \bmod n$$

相应的解密算法：

$$d_K(y) = y^a \bmod n$$

由于 $n=pq$，所以有：

$$\varphi(n) = (p-1)(q-1)$$

可对解密算法进行简单地验证：

$$\begin{aligned}
& y^a \bmod n \\
&= (x^b)^a \bmod n \\
&\equiv x^{t\varphi(n)+1} (\bmod n) \\
&\equiv (x^{\varphi(n)})^t x (\bmod n) \\
&\equiv x (\bmod n)
\end{aligned}$$

以上的验证说明通过解密可以还原出明文。

例 3.8　在 RSA 公钥密码系统中，已知 $p=499$、$q=929$，则：

$$n = 499 \times 929 = 463\,571$$

$$\varphi(n) = (499-1)(929-1) = 462\,144$$

取：

$$a = 255\,157, \quad b = 9949$$

对明文 $x=200\,412$ 实施 RSA 加密算法：

$$y = 200\,412^{9949} \bmod 463\,571 = 418\,883$$

实施 RSA 解密算法：

$$x = 418\,883^{255\,157} \bmod 463\,571 = 200\,412$$

下面对 RSA 的安全性进行分析。分析之前，首先明确 NP 问题的概念。

一个问题的复杂性建立在问题求解算法的复杂性基础之上，通常可以将问题的求解算法简单分为以下两类：

(1) 确定性算法。确定性算法的每一步操作，其结果是确定的，算法所用的时间是这些确定步骤的时间总和。

(2) 非确定性算法。非确定性算法的某些操作结果是不确定的，这样算法所用的时间也是不确定的。其中，导致成功的步骤所需的最少时间就是非确定性算法的计算时间。

用确定性算法可以在多项式时间内求解的问题称为确定性多项式时间可解类,或简称P问题。用非确定性算法可以在多项式时间内求解的问题称为非确定性多项式时间可解类,或简称NP完全问题或NP问题。

显然,P⊆NP,也就是说NP中的许多问题要比P中的问题更难,但是目前依然没有人能够证明:P≠NP。

对于NP问题,不存在任何已知的确定性算法在多项式时间内求解该问题,也就是说,NP被认为是计算上足够难的问题,于是NP问题常在密码学中出现,用以构造公钥密码体制。

$n=pq$ 的大数因式分解问题就被广大学者认为是一个NP问题。

在RSA公钥密码体制中,攻击者希望通过公钥 n 得到私钥 p 和 q,但这是一个NP问题。既然NP问题难以求解,那么有没有避开大数因式分解NP问题而破解RSA公钥密码系统的可能呢?

由于 n,b 是公钥,私钥 b 和公钥 a 有以下联系:

$$ab \equiv 1(\bmod\ \varphi(n))$$

所以,如果能得到 $\varphi(n)$将是避开大数因式分解问题的最佳办法。但是,下面的定理将说明求 $\varphi(n)$与大数因式分解等价。

定理 3.7 大数 $n=pq$ 的因式分解等价于求 $\varphi(n)$。

证明:$n=pq$,如果 p 和 q 已求出,那么:

$$\varphi(n) = (p-1)(q-1)$$

反过来,如果已知 $\varphi(n)$,则:

$$\varphi(n) = (p-1)(q-1) = pq - p - q + 1 = n - (p+q) + 1$$

于是有:

$$p + q = n - \varphi(n) + 1$$

结合 $pq=n$ 得:

$$p - q = \sqrt{(p+q)^2 - 4pq} = \sqrt{(p+q)^2 - 4n}$$

联合以上两个式子,可得:

$$p = (n - \varphi(n) + 1 + \sqrt{(p+q)^2 - 4n})/2$$

$$q = (n - \varphi(n) + 1 - \sqrt{(p+q)^2 - 4n})/2$$

命题得证。

从上面的定理可以看到求 $\varphi(n)$和大数因式分解是等价的,也就是说,求 $\varphi(n)$同大数因式分解同样困难。

虽然不可能通过求 $\varphi(n)$来破解因式分解问题,但是避免大数因式分解问题直接攻击RSA的可能性依然存在,下面就是一个例子。

例 3.9 在RSA公钥密码系统中,已知 $p=23$、$q=31$,那么:

$$n = 23 \times 31 = 713$$

$$\varphi(n) = (23-1)(31-1) = 660$$

取:

$$a = 569, \quad b = 29$$

对明文 $x=25$ 进行加密：

$$y(0) = 25^{29} \bmod 713 = 36$$

在不知道私钥 a 的情况下，再进行几遍加密：

$$y(1) = 36^{29} \bmod 713 = 676$$
$$y(2) = 676^{29} \bmod 713 = 625$$
$$y(3) = 625^{29} \bmod 713 = 583$$
$$y(4) = 583^{29} \bmod 713 = 656$$
$$y(5) = 656^{29} \bmod 713 = 614$$
$$y(6) = 614^{29} \bmod 713 = 501$$
$$y(7) = 501^{29} \bmod 713 = 397$$
$$y(8) = 397^{29} \bmod 713 = 532$$
$$y(9) = 532^{29} \bmod 713 = 25$$
$$y(10) = 25^{29} \bmod 713 = 36$$

于是，通过若干次的连续加密后，明文被恢复出来，RSA 算法受到攻击。通常将这种攻击方式称为循环攻击。

究竟什么情况能产生循环攻击呢？

反复进行 RSA 加密的序列可表示成：

$$x^b \bmod n$$
$$(x^b)^b \bmod n$$
$$\cdots$$
$$((x^b)^b \cdots)^b \bmod n$$

令：

$$((x^b)^b \cdots)^b = x \bmod n$$

即：

$$x^{b^k} = x \bmod n \tag{3.1}$$

因为 $n=pq$，所以式(3.1)可分解称同余方程组：

$$\begin{cases} x^{b^k} = x \bmod p \\ x^{b^k} = x \bmod q \end{cases} \tag{3.2}$$

因为 p、q 为素数，假设 $x \neq p$、$x \neq q$，所以：

$$\begin{cases} x^{\varphi(p)} = 1 \bmod p \\ x^{\varphi(q)} = 1 \bmod q \end{cases} \tag{3.3}$$

结合式(3.1)和式(3.2)可得：

$$\varphi(p) \mid (b^k - 1)$$
$$\varphi(q) \mid (b^k - 1)$$

显然，这就是产生循环攻击可能性的条件。

通过验证，例 3.9 正好满足此条件：

$$\varphi(23) = 22, \quad \varphi(30) = 8$$
$$29^{10} = 420\,707\,233\,300\,201$$

$$22 \mid (420\ 707\ 233\ 300\ 201 - 1)$$
$$8 \mid (420\ 707\ 233\ 300\ 201 - 1)$$

循环攻击是一种典型的仅知密文攻击，除循环攻击之外，对 RSA 公钥密码系统的常见攻击方式还有同模攻击。

同模攻击：假设两个 RSA 公钥密码系统共享同一个 n，但参数 b 各不相同，分别为 b_1 和 b_2，攻击者已获得同一个明文对应的两个密文 y_1 和 y_2，那么攻击者将能够得到明文 x，计算步骤如下：

(1) 计算 $c_1 = b_1^{-1} \bmod b_2$；

(2) 计算 $c_2 = (c_1 b_1 - 1)/b_2 \bmod n$；

(3) 计算 $x = y_1^{c_1} (y_2^{c_2})^{-1} \bmod n$。

例 3.10 在 RSA 公钥密码系统中，已知 $p=89$，$q=137$，$n=89\times137=12\ 193$。取：

$$a_1 = 10\ 429,\quad b_1 = 661$$
$$a_2 = 4468,\quad b_2 = 1181$$

分别对明文 $x=2005$ 进行加密：

$$y_1 = 2005^{661} \bmod 12\ 193 = 3429$$
$$y_2 = 2005^{1181} \bmod 12\ 193 = 11\ 196$$

实施同模攻击算法：

$$c_1 = 661^{-1} \bmod 1181 = 1047$$
$$c_2 = (1047 \times 661 - 1)/1181 \bmod 12\ 193 = 586$$
$$x = 3429^{1047} (11\ 196^{586})^{-1} \bmod 12\ 193 = 2005$$

同模攻击的前提是有相同的 n 和相同的明文。在实际应用中，明文相同是经常发生的。要避免同模攻击，关键是应在设计密钥时避免不同的通信中使用相同的 n。

无论是同模攻击还是循环攻击，对 RSA 算法并不能构成真正的威胁，因为它们的攻击都是建立在参数选取不当的基础之上。

3.3 RSA 实现的问题

RSA 是一个安全性和实用性都很强的公钥密码体制。不过，在算法具体实现过程中，仍需要考虑以下两个关键问题：

(1) 如何找到足够大的素数？

(2) 如何高效地实现模指数运算？

下面围绕这两个问题进行讨论。

要得到一个足够大的素数，无非有两类方法。一类是通过数学理论构造出一个大的素数，另一类是随机给定一个足够大的数，然后去证明它是素数。本节主要介绍第二类方法。

验证一个数是不是素数，又称为**素性检测**，素性检测可以用威尔逊(Wilson)定理。

定理 3.8 (Wilson 定理) p 是素数 $\Leftrightarrow (p-1)! \equiv -1 \bmod p$。

证明：已知 $(p-1)! \equiv -1 \bmod p$，若 p 不是素数，不妨设 $p=ab$，$1<a<p$，$1<b<p$，

显然：

$$a \mid (p-1)!$$

因为$(p-1)! \equiv -1 \bmod p$，所以：

$$p \mid (p-1)!+1$$
$$\Rightarrow a \mid (p-1)!+1$$
$$\Rightarrow a \mid 1$$

产生矛盾，所以 p 必为素数。

反过来，如果 p 是素数，当 p 取 2、3 时，容易验证$(p-1)! \equiv -1 \bmod p$，命题成立；当 $p>3$ 时，考虑集合 $S=\{2,3,\cdots,p-2\}$中的任意元素 a，因为$(a,p)=1$，有整数 k_1 和 k_2 使得：

$$k_1 a + k_2 p = 1$$
$$k_1 a \equiv 1 \pmod p$$

于是，存在 b 是 a 的逆元，$1<b<p-1$，使得 $b \equiv k_1 \pmod p$。

接下来证明 $b \neq a$：

如果 $b=a$，则 $a^2 \equiv 1 \pmod p$，a 要么等于 1，要么等于 $p-1$，都与 $a \in S$ 产生矛盾，所以 $b \neq a$。

取 $a' \in S$ 且 $a' \neq a, a' \neq b$，则有：

$$a'b' = 1 \pmod p$$

且：

$$b' \neq a', \quad b' \neq a, \quad b' \neq b$$

于是 S 中的数可以分成$(p-3)/2$ 对，每一对数互逆，因此：

$$2 \times 3 \times \cdots \times (p-2) \equiv 1 \pmod p$$
$$(p-1)! = (p-1) = -1 \pmod p$$

命题得证。

例 3.11　因为

$$22! = 1124000727777607680000 = -1 \bmod 23$$

根据 Wilson 定理，可得 23 是素数。

Wilson 定理虽然能判定一个整数是否是素数，但是这个算法的运算量非常大，并不实用。

为了寻求更高效的素性检测方法，下面先来回顾一下费马定理。

由 Fermat 定理可知，若 n 为素数，则对任意与 n 互素的整数 b，有：

$$b^{n-1} = 1 \pmod n$$

Fermat 定理说明：上式是 n 为素数的必要条件。

将 Fermat 定理变成其逆反命题，可得：如果 b 和 n 互素，且：

$$b^{n-1} \neq 1 \pmod n$$

那么 n 一定不是素数，而是一个合数。

反过来，如果成立：

$$b^{n-1} = 1 \pmod n$$

那么能判定 n 一定是素数吗？答案是不一定。

例如,(2,63)=1,由 $2^{62}=(2^6)^{10}2^2=4\neq1(\bmod\ 63)$ 可知 63 是合数;

又如,(8,63)=1,且 $8^{62}=(8^2)^{31}=1(\bmod\ 63)$,但是 63 却不是素数。

进一步,可引入"拟素数"的概念。

拟素数:设 n 是一个奇合数,若整数 b 与 n 互素,成立

$$b^{n-1}=1(\bmod\ n)$$

则 n 称为对于基 b 的拟素数。

例 3.12 整数 63 是对于基 $b=8$ 的拟素数。

按照统计规律,拟素数有一个重要性质:

随机选取与 n 互素的整数 b,若 $b^{n-1}=1(\bmod\ n)$ 成立,则 n 为拟素数的概率 $\leqslant50\%$,n 为素数的概率 $\geqslant50\%$。

该性质可以用来构造素性检测算法,算法如下:

给定奇整数 $n\geqslant3$ 和检测轮数 t,令 $k=1$。

(1) 随机选取整数 b,$2\leqslant b\leqslant n-2$。

(2) 计算 $d=(b,n)$。

(3) 若 $d>1$,则 n 是合数,输出"Fail",结束;否则转(4)。

(4) 计算 $r=b^{(n-1)/2}(\bmod\ n)$。

(5) 若 $r\neq\pm1(\bmod\ n)$,则 n 是合数,输出"Fail",结束;否则,表明 n 通过了一轮素性检测,转到(6)。

(6) $k=k+1$。若 $k\leqslant t$,转到(1);否则表明 n 通过了 t 轮检测,输出"Success",结束。

显然,经过 t 轮检测并输出"Success"的拟素数确定为素数的概率是 $1-0.5^t$。然而,这种概率素性检测算法的应用很快便受到卡密歇尔数的挑战。

卡密歇尔(Carmichael)数:合数 n 称为卡密歇尔数,如果对所有与 n 互素的正整数 b 都满足:

$$b^{n-1}=1(\bmod\ n)$$

例如,整数 $561=3\times11\times17$ 是一个卡密歇尔数。

又如,整数 $1105=5\times13\times17$ 也是一个卡密歇尔数。

如果卡密歇尔数是有限的,那么前面的概率素性检测算法还是可用的,只需在每次算法运行前先查询"卡密歇尔数表"即可。但遗憾的是事实上关于卡密歇尔数有以下两条重要结论:

(1) 存在无穷多个卡密歇尔数;

(2) 当 n 足够大时,区间 $[2,n]$ 内至少有 $n^{2/7}$ 个卡密歇尔数。

虽然前面的拟素数的性质难以构造出可靠的概率素性检测算法,但接下来的 Euler 拟素数却能引出切实可行的概率素性检测算法。

欧拉(Euler)拟素数:已知 n 是奇素数,如果对任意正整数 a,成立

$$\left(\frac{a}{n}\right)=a^{(n-1)/2}(\bmod\ n)$$

那么称 n 是对于基 a 的欧拉拟素数。

之所以 n 被称为欧拉拟素数,来源于二次剩余的欧拉准则。

二次剩余:设 n 是奇素数,$a=0(\bmod\ n)$,$a\neq0(\bmod\ b)$,若方程 $x^2=a(\bmod\ n)$ 有解,则

称 a 是模 n 的平方剩余。

欧拉准则：设 n 是奇素数，a 是整数，那么 a 是一个模 n 的二次剩余当且仅当

$$a^{(n-1)/2} = 1(\mathrm{mod}\ n)$$

假定 n 是一个奇素数。对于任意整数 a，勒让德(Legendre)符号 $\left(\frac{a}{n}\right)$ 定义如下：

$$\left(\frac{a}{n}\right)=\begin{cases}0 & a = 0(\mathrm{mod}\ n)\\ 1 & a\text{ 是模 }n\text{ 的平方剩余}\\ -1 & a\text{ 不是模 }n\text{ 的平方剩余}\end{cases}$$

下面的定理显然成立。

定理 3.9　当 n 是奇素数时，等式：

$$\left(\frac{a}{n}\right)= a^{(n-1)/2}(\mathrm{mod}\ n)$$

成立。

反过来，如果等式成立，是否就意味着 n 是素数呢？答案是不一定。因此，称满足等式的 n 是欧拉拟素数。

雅可比符号：设 n 是正奇数，且 n 的因式分解可写成：

$$n = \prod_{i=1}^{k} p_i^{e_i}$$

设 a 为一个整数，那么雅可比(Jacobi)符号定义为：

$$\left(\frac{a}{n}\right)= \prod_{i=1}^{k}\left(\frac{a}{p_i}\right)^{e_i}$$

当 n 是素数时，雅可比符号与勒让德符号是一样的。

为判定一个数是否是欧拉拟素数，下面给出雅可比符号的计算方法。

雅可比符号的计算方法可由雅可比符号的 4 个性质确定：

(1) 设 $m_1=m_2(\mathrm{mod}\ n)$，那么 $\left(\frac{m_1}{n}\right)=\left(\frac{m_2}{n}\right)$；

(2) $\left(\frac{2}{n}\right)=\begin{cases}1 & n=\pm1(\mathrm{mod}\ 8)\\ -1 & n=\pm3(\mathrm{mod}\ 8)\end{cases}$；

(3) $\left(\frac{m_1m_2}{n}\right)=\left(\frac{m_1}{n}\right)\left(\frac{m_2}{n}\right)$；

(4) 设 m 是正奇数，$\left(\frac{m}{n}\right)=\begin{cases}-\left(\frac{n}{m}\right) & m=n=3(\mathrm{mod}\ 4)\\ \left(\frac{n}{m}\right) & \text{其他情况}\end{cases}$。

例 3.13　计算 $\left(\frac{6278}{9975}\right)$。

解：因为 $9975=3\times5^2\times7\times19$

$$\begin{aligned}\left(\frac{6278}{9975}\right)&= \left(\frac{6278}{3}\right)\left(\frac{6278}{5}\right)^2\left(\frac{6278}{7}\right)\left(\frac{6278}{19}\right)\\&= \left(\frac{2}{3}\right)\left(\frac{3}{5}\right)^2\left(\frac{6}{7}\right)\left(\frac{8}{19}\right)\end{aligned}$$

$$=(-1)(-1)^2(-1)(-1)$$
$$=-1$$

例 3.14 91 是基 10 的欧拉拟素数，这是因为：

$$10^{(91-1)/2}=-1(\text{mod } 91)$$
$$\left(\frac{10}{91}\right)=\left(\frac{10}{7}\right)\left(\frac{10}{13}\right)=\left(\frac{3}{7}\right)\left(\frac{2}{13}\right)\left(\frac{5}{13}\right)=-\left(\frac{7}{3}\right)(-1)\left(\frac{13}{5}\right)$$
$$=\left(\frac{1}{3}\right)\left(\frac{3}{5}\right)=-\left(\frac{5}{3}\right)=\left(\frac{2}{3}\right)=-1$$
$$\left(\frac{10}{91}\right)=10^{(91-1)/2}(\text{mod } 91)$$

但是 91 实际上是合数。

此处不加证明地给出类似于拟素数的结论：欧拉拟素数是素数的概率≥50%。

利用这一结论可以构造出 Solovay-Stassen 概率素性检测算法。

Solovay-Stassen 概率素性检测算法：

给定奇整数 $n\geq 3$ 和检测轮数 t，令 $k=1$。

(1) 随机选取整数 a，$2\leq a\leq n-2$。

(2) 计算 $d=\gcd(a, n)$。

(3) 判断：若 $d>1$，则 n 是合数，输出“Fail”，结束；否则转到(4)。

(4) 计算 $r=a^{(n-1)/2}(\text{mod } n)$。

(5) 判断：若 $r\neq\pm 1(\text{mod } n)$，则 n 是合数，输出“Fail”，结束；否则转(6)。

(6) 计算勒让德符号 $s=\left(\frac{a}{n}\right)$。

(7) 判断：若 $r\neq s$，则 n 是合数，输出“Fail”，结束；否则转(8)。

(8) n 通过一轮素性检测，$k=k+1$。

(9) 若 $k\leq t$，转(1)，否则 n 通过 t 轮检测，输出“Success”，结束。

通过 t 次 Solovay-Stassen 素性检测的整数 n 为素数的概率：

$$P\{n \text{ 为素数}\}\geq 1-0.5^t$$

与拟素数不同，卡密歇尔数对欧拉拟素数不构成任何威胁，Solovay-Stassen 概率素性检测算法具有较强的实用性。

强拟素数：设 n 为正奇合数，且 $n-1=2^s d$（d 为奇数），a 与 n 互素，如果有：

$$a^d=1(\text{mod } n)$$

或存在 $r(0\leq r<s)$，使得：

$$a^{2^r d}=-1(\text{mod } n)$$

则称 n 为对于基 a 的强拟素数。

强拟素数也与欧拉定理有关，具体原理如下：

设 n 是奇整数，且 $n-1=2^S d$，则有：

$$a^{n-1}-1=a^{2^s d}-1=(a^d)^{2^s}-1$$
$$=((a^d)^{2^{s-1}}+1)((a^d)^{2^{s-1}}-1)=(a^{2^{s-1}d}+1)(a^{2^{s-1}d}-1)$$
$$=(a^{2^{s-1}d}+1)(a^{2^{s-2}d}+1)(a^{2^{s-2}d}-1)$$
$$=\cdots$$

$$= (a^{2^{s-1}d}+1)(a^{2^{s-2}d}+1)\cdots(a^{d}+1)(a^{d}-1)$$

因此，若有：

$$a^{n-1} = 1(\bmod n)\text{，即 } a^{n-1}-1 = 0(\bmod n)$$

即：

$$(a^{2^{s-1}d}+1)(a^{2^{s-2}d}+1)\cdots(a^{d}+1)(a^{d}-1) = 0(\bmod n)$$

于是下列等式中至少有一个成立：

$$a^{d} = 1(\bmod n)$$
$$a^{d} = -1(\bmod n)$$
$$a^{2d} = -1(\bmod n)$$
$$a^{2^2 d} = -1(\bmod n)$$
$$\cdots$$
$$a^{2^{s-1}d} = -1(\bmod n)$$

例 3.15　整数 $n=2047=23\times 89$ 是对于基 $a=2$ 的强拟素数。

这是因为：

$$n-1 = 2047-1 = 2046 = 2\times 1023$$
$$2^{1023} = (2^{11})^{93} = 2048^{93} = 1(\bmod 2047)$$

而实际上，2047 并不是素数，因为 $2047=23\times 89$。2047 是一个强拟素数。

此处不加证明地给出关于强拟素数概率的结论：

设 n 是一个奇合数，则 n 是对于基 b 的强拟素数的可能性≤25%；于是可以利用强拟素数构造新的概率素性检测算法。

Miller-Rabin 素性检测算法：给定奇整数 $n\geqslant 3$ 和检测轮数 k，令变量 $i=1$，设 $n-1=2^{s}d$，其中 d 为奇数。

(1) 随机选取整数 b，$2\leqslant a\leqslant n-2$。

(2) 计算 $k=(a, n)$。

(3) 判断：若 $k>1$，转到(12)；否则转到(4)。

(4) 令 $j=0$，计算 $r=a^{d}(\bmod n)$。

(5) 判断：若 $r=\pm 1(\bmod n)$，转到(10)；否则，转到(6)。

(6) $j=j+1$。

(7) 若 $j<s$，转到(8)；否则，转到(12)。

(8) 计算 $r=r^{2}(\bmod n)$。

(9) 判断：若 $r=-1(\bmod n)$，转到(10)；否则转到(6)。

(10) (n 通过一轮素性检测)，$i=i+1$。

(11) 若 $i\leqslant k$，转到(1)，否则 n 通过 k 轮检测，输出“Success”，转到(13)。

(12) n 是合数(n 未通过本轮检测)，输出“Fail”，转到(13)。

(13) 结束。

通过 t 次 Miller-Rabin 素性检测的整数 n 为素数的概率：

$$P\{n \text{ 为素数}\} \geqslant 1-0.25^{t}$$

由此可见，与 Solovay-Stassen 素性检测算法相比，Miller-Rabin 素性检测算法的检测效

率更高一些。

例 3.16 试判断 162 817 是否是素数。

解：由 $n=162\,817$ 得：

$$n-1=162\,817-1=162\,816=2^{10}\times 159$$

选基 $a=2,t=2$。第 1 轮计算：

$$j=0:r_0=2^{159}=121\,518\neq\pm 1(\text{mod }162\,817)$$
$$j=1:r_1=r_0^2=2^{318}=121\,518^2=99\,326\neq -1(\text{mod }162\,817)$$
$$j=2:r_2=r_1^2=2^{636}=99\,326^2=83\,795\neq -1(\text{mod }162\,817)$$
$$j=3:r_3=r_2^2=2^{1272}=83\,795^2=11\,890\neq -1(\text{mod }162\,817)$$
$$j=4:r_4=r_3^2=2^{2544}=11\,890^2=135\,524\neq -1(\text{mod }162\,817)$$
$$j=5:r_5=r_4^2=2^{5088}=135\,524^2=20\,074\neq -1(\text{mod }162\,817)$$
$$j=6:r_6=r_5^2=2^{10\,176}=20\,074^2=156\,218\neq -1(\text{mod }162\,817)$$
$$j=7:r_7=r_6^2=2^{20\,352}=156\,218^2=74\,662\neq -1(\text{mod }162\,817)$$
$$j=8:r_8=r_7^2=2^{40\,704}=74\,662^2=48\,615\neq -1(\text{mod }162\,817)$$
$$j=9:r_9=r_8^2=2^{81\,408}=48\,615^2=129\,470\neq -1(\text{mod }162\,817)$$

第 1 轮素性检测未通过。

第 2 轮计算：

$$j=0:r_0=3^{159}=14\,319\neq\pm 1(\text{mod }162\,817)$$
$$j=1:r_1=r_0^2=3^{318}=14\,319^2=47\,158\neq -1(\text{mod }162\,817)$$
$$j=2:r_2=r_1^2=3^{636}=47\,158^2=122\,378\neq -1(\text{mod }162\,817)$$
$$j=3:r_3=r_2^2=3^{1272}=122\,378^2=141\,590\neq -1(\text{mod }162\,817)$$
$$j=4:r_4=r_3^2=3^{2544}=141\,590^2=70\,890\neq -1(\text{mod }162\,817)$$
$$j=5:r_5=r_4^2=3^{5088}=70\,890^2=45\,395\neq -1(\text{mod }162\,817)$$
$$j=6:r_6=r_5^2=3^{10\,176}=45\,395^2=94\,073\neq -1(\text{mod }162\,817)$$
$$j=7:r_7=r_6^2=3^{20\,352}=94\,073^2=136\,928\neq -1(\text{mod }162\,817)$$
$$j=8:r_8=r_7^2=3^{40\,704}=136\,928^2=85\,549\neq -1(\text{mod }162\,817)$$
$$j=9:r_9=r_8^2=3^{81\,408}=85\,549^2=7251\neq -1(\text{mod }162\,817)$$

第 2 轮素性检测也未通过。

于是，162 817 是素数的概率是：

$$1-0.25^2=0.9375$$

前面给出了概率素性检测的几种算法，尤其是 Miller-Rabin 素性检测算法能够提供非常有效的鉴别素数的手段。但是，寻找一个大素数和素性检测并不是完全等同的，它们之间相差一个素数存在性问题。是否会出现这种可能性：尽管进行了大批量的素数检测，但始终找不出一个理想的素数？

好在素数定理能够给出从 1 到 N 的整数中抽到素数的概率：

$$\Pi(N)\approx N/\text{In}N$$

于是，可以进一步得出：从不大于 N 的正整数中随机选一个正整数，它是素数的概率大约是：

$$(N/\text{In}N)/N = 1/\text{In}N$$

例如，任选一个 512 位的随机正整数，它是素数的概率大约为：

$$1/\text{In}2^{512} = 1/177$$

即平均 177 个 512 位以内的随机正整数中将有一个素数。

除寻找大素数外，RSA 算法实现中还需处理好另一个问题，即模指数运算问题。因为无论加密还是解密都需要用到模指数运算，模指数运算的效率是影响 RSA 算法效率的重要因素。

常用的模指数算法有二进制法、二进制 NAF 算法、滑动窗口算法等。下面重点介绍二进制法。

二进制法是指将指数二进制化，即指数写成二进制的形式：

$$e = (e_l \quad e_{l-1} \quad \cdots \quad e_1 \quad e_0)_2 \quad e_i \in \{0,1\}$$

那么模指数运算：

$$g^e \bmod n = (((g^{e_l})^2 \times g^{e_{l-1}})^2 \times \cdots \times g^{e_1})^2 \times g^{e_0}$$

根据以上原理，可形成以下具体算法(Right to Left Binary Method)：

```
s = 1
for i = l to 0 do {
s = s × s mod n
if(eᵢ = 1)   then s = s × g mod n
}
return s
```

令 $w(e)$ 是二进制表示中 1 的个数，以上算法要做 $l-1$ 次平方和 $w(e)-1$ 次乘法运算。如果 e 是在区间 $0<e<n$ 上随机选取，则需要大约 $\lg(n)$ 次平方和 $0.5\times(\lg(n)+1)$ 次乘法，效率较高。

例 3.17　计算 $9726^{3533} \bmod 11\,413$。

解：3533 转化为二进制是 110111001101。

运算过程如下：

$$i = 11: b_i = 1, \quad s = 1^2 \times 9726 = 9726$$
$$i = 10: b_i = 1, \quad s = 9726^2 \times 9726 = 2659$$
$$i = 9: b_i = 0, \quad s = 2659^2 = 5634$$
$$i = 8: b_i = 1, \quad s = 5634^2 \times 9726 = 9167$$
$$i = 7: b_i = 1, \quad s = 9167^2 \times 9726 = 4958$$
$$i = 6: b_i = 1, \quad s = 4958^2 \times 9726 = 7783$$
$$i = 5: b_i = 0, \quad s = 7783^2 = 6298$$
$$i = 4: b_i = 0, \quad s = 6298^2 = 4629$$
$$i = 3: b_i = 1, \quad s = 4629^2 \times 9726 = 10\,185$$
$$i = 2: b_i = 1, \quad s = 10\,185^2 \times 9726 = 105$$
$$i = 1: b_i = 0, \quad s = 105^2 = 11\,025$$
$$i = 0: b_i = 1, \quad s = 11\,025^2 \times 9726 = 5761$$

于是得到结果：

$$9726^{3533} \bmod 11\,413 = 5761$$

3.4 大数因式分解问题

对 RSA 算法真正的威胁来自于大数因式分解问题的求解。目前，已有一些大数因式分解问题的解决方案，如 $p-1$ 算法、Dixon 算法等。

$p-1$ 算法由 Pollard 于 1974 年提出。具体流程如下所述：

```
a = 2
for i = 2 to b
        a = a^i mod n
d = gcd(a - 1, n)
if(1 < d < n) then
        d 是 n 的素因子
else
        寻找 n 的素因子失败
```

$p-1$ 算法有两个输入，一个是要分解的数 n，一个是边界值 b。

假设 p 是 n 的一个素数因子，如果对每一个素数 $q|(p-1)$，有 $q \leqslant b$，则必有：

$$(p-1) \mid b! \tag{3.4}$$

在循环结束时有：

$$a = 2^{b!} \bmod n$$

于是：

$$a = 2^{b!} \bmod p \tag{3.5}$$

根据欧拉定理可得：

$$2^{p-1} = 1 \bmod p \tag{3.6}$$

由式(3.3)、式(3.4)和式(3.5)得：

$$a = 1 \bmod p$$

所以：

$$p \mid d = \gcd(a-1, n)$$

例 3.18 设 $n=29\ 389\ 613\ 454\ 601$，取 $b=600$，运用 $p-1$ 算法进行因式分解。在循环结束时，得到：

$$a = 22\ 058\ 222\ 139\ 834$$

于是：

$$d = \gcd(22\ 058\ 222\ 139\ 833,\ 29\ 389\ 613\ 454\ 601) = 6\ 210\ 433$$

进一步检测：

$$29\ 389\ 613\ 454\ 601 = 6\ 210\ 433 \times 4\ 732\ 297$$

说明因式分解成功。

同样的例子，如果 b 选取不当，如 $b=200$，在循环结束时，得到：

$$a = 8\ 688\ 934\ 735\ 738$$

于是：

$$d = \gcd(8\ 688\ 934\ 735\ 737,\ 29\ 389\ 613\ 454\ 601) = 1$$

说明在这种情况下因式分解失败。

Dixon 算法建立在一个简单的事实基础上：

如果 $x \neq \pm y \pmod n$ 且 $x^2 = y^2 \pmod n$，那么 $\gcd(x-y, n)$ 是 n 的非平凡因子(Non-Trivial Factor)。

该算法首先建立一个小素数集合 B，然后找出一些整数 x 使得 $x^2 \bmod n$ 的所有素因子都在因子集合 B 之中。接下来，将某些 x 相乘使得每一个素数出现偶数次，于是就形成了同余方程：

$$x^2 = y^2 \pmod n$$

例 3.19　设 $n=52\,487\,693$，令 $B=\{3,5,13\}$，以下等式成立：

$$14\,071\,835^2 = 3 \times 7 \pmod{52\,487\,693}$$
$$35\,191\,901^2 = 3 \times 13 \pmod{52\,487\,693}$$
$$19\,785\,682^2 = 7 \times 13 \pmod{52\,487\,693}$$

将上面 3 个式子相互乘起来：

$$(14\,071\,835 \times 35\,191\,901 \times 19\,785\,682)^2 = (3 \times 7 \times 13)^2 \pmod{52\,487\,693}$$

即：

$$43\,632\,496^2 = 273^2 \pmod{52\,487\,693}$$

于是：

$$\gcd(43\,632\,496 - 27\,315\,770\,708\,441) = 4007$$

得到了 n 的一个因子是 4007。

因为 4007 是一个素数，所以另一个因子是：

$$52\,487\,693/4007 = 13\,099$$

写成因式分解的形式是：

$$52\,487\,693 = 4007 \times 13\,099$$

3.5　Rabin 公钥密码系统

RSA 的安全性建立在大数因式分解 NP 问题基础之上，那么是否可以在大数因式分解 NP 问题上构造出不同于 RSA 的公钥密码系统呢？答案是肯定的，Rabin 公钥密码系统就是一例。

在介绍 Rabin 公钥密码系统之前，先介绍同余方程组。

《孙子算经》中曾经提出一个问题："今有物不知其数，三三数之剩二，五五数之剩三，七七数之剩二，问物几何？"

这是一个典型的同余方程组求解的问题，描述如下：

$$\begin{cases} x = a_1 \bmod m_1 \\ x = a_2 \bmod m_2 \\ \vdots \\ x = a_n \bmod m_n \end{cases} \tag{3.7}$$

同余方程组问题通常由中国剩余定理求解，中国剩余定理也称孙子定理。

定理 3.10 （中国剩余定理）设 m_1、m_2、…、m_n 是 $n(n\geqslant 2)$个两两互素的大于 1 的整数，令：

$$M = m_1m_2\cdots m_n = m_1M_1 = m_2M_2 = \cdots = m_nM_n$$

则同余方程组的解为：

$$x \equiv a_1b_1M_1 + a_2b_2M_2 + \cdots + a_nb_nM_n(\bmod M) \tag{3.8}$$

其中，$b_i(1\leqslant i\leqslant n)$是满足同余方程

$$M_ix_i \equiv 1(\bmod m_i) \tag{3.9}$$

的一个特解。

证明：可以分成以下两步来证明：

(1) 证明式(3.8)是同余方程组的解。将式(3.7)代入同余方程组中，对于任意第 i 个方程，等式左边等于：

$$x \equiv a_1b_1M_1 + a_2b_2M_2 + \cdots + a_nb_nM_n(\bmod M)$$

其中，除第 i 项外其余各项都能被 m_i 整除，而第 i 项为 $a_ib_iM_i$，由式(3.8)可得：

$$a_ib_iM_i \equiv a_i(\bmod m_i)$$

(2) 证明同余方程组的解均在式(3.7)中。设 x'是同余方程组中的任意一个解：

$$x' \equiv a_i(\bmod m_i),\quad i = (1,2,\cdots,n)$$

设 x 是由式(3.7)和式(3.8)确定的解，则：

$$x - x' \equiv 0(\bmod m_i),\quad i = (1,2,\cdots,n)$$

$$m_i \mid (x - x'),\quad i = (1,2,\cdots,n)$$

又因为 m_1、m_2、…、m_n 两两互素，所以：

$$m_1m_2\cdots m_n \mid (x - x')$$

$$x' \equiv x(\bmod M)$$

也就是说，x'被包含于由式(3.7)和式(3.8)确定的解中。

命题得证。显然，在同余方程组的最小非负整数解是 x 对模 M 的主余数。

此外，中国剩余定理还有另一种表述："设 m_1、m_2、…、m_n 是 $n(n\geqslant 2)$个两两互素的大于 1 的整数，令 $M=m_1m_2\cdots m_n$，则对于任意的 $j(1\leqslant j\leqslant n)$，以下联立同余方程组有解：

$$x_j \equiv 1 \bmod m_j$$

$$x_j \equiv 0 \bmod m_i, i \neq j$$

令 $x=a_1x_1+a_2x_2+\cdots+a_nx_n$，则$[x]_m$ 是同余方程组的解。"

只需证明 $x_i=b_iM_i$ 就可说明两种表述的等价性，感兴趣的读者可将此命题的证明作为习题。

例 3.20 求解 x，x 满足下列同余方程组：

$$\begin{cases} x \equiv 5(\bmod 7) \\ x \equiv 3(\bmod 11) \\ x \equiv 10(\bmod 13) \end{cases}$$

解：

$$a_1 = 5,\quad a_2 = 3,\quad a_3 = 10$$

$$m_1 = 7,\quad m_2 = 11,\quad m_3 = 13$$

$$M = m_1m_2m_3 = 1001$$

$$M_1 = m_2 m_3 = 143,\quad M_2 = m_1 m_3 = 91,\quad M_3 = m_1 m_2 = 77$$

可通过欧几里德扩展算法来求解 b_i：

$$b_1 = 143^{-1} = 5(\bmod 7)$$

$$b_2 = 91^{-1} = 4(\bmod 11)$$

$$b_3 = 77^{-1} = 12(\bmod 13)$$

该同余方程组最终的解为：

$$x = 13\,907 = 894(\bmod 1001)$$

中国剩余定理不仅可以用来求解同余方程组，还可以用来求解二次同余方程，其方法就是将二次同余方程化为同余方程组来求解。

例 3.21　求解二次同余方程：$x^2+3x+2\equiv 0(\bmod 35)$。

解：可将二次同余方程变为同余方程组：

$$\begin{cases} x^2 + 3x + 2 \equiv 0(\bmod 5) \\ x^2 + 3x + 2 \equiv 0(\bmod 7) \end{cases} \tag{3.10}$$

因为 $x^2+3x+2=(x+1)(x+2)$，所以进一步可将二次同余方程组(3.10)分解为 4 个一阶同余方程组：

$$\begin{cases} x \equiv 3(\bmod 5) \\ x \equiv 5(\bmod 7) \end{cases} \qquad \begin{cases} x \equiv 3(\bmod 5) \\ x \equiv 6(\bmod 7) \end{cases}$$

$$\begin{cases} x \equiv 4(\bmod 5) \\ x \equiv 5(\bmod 7) \end{cases} \qquad \begin{cases} x \equiv 4(\bmod 5) \\ x \equiv 6(\bmod 7) \end{cases}$$

根据中国剩余定理可分别求出 4 个同余方程组的解：

$$x \equiv 33,13,19,34(\bmod 35)$$

Rabin 公钥密码系统：设 $n=pq$ 是 p 和 q 两个互不相同的大素数的乘积，且

$$p,q \equiv 3(\bmod 4)$$

明文空间和密文空间：

$$P = C = Z_n$$

密钥：

$$K = \{(n,p,q,B)\}$$

其中，$0\leqslant B\leqslant n-1$。

加密算法：

$$e_K(x) = x(x+B) \bmod n$$

解密算法：

$$d_K(y) = \left(\sqrt{\frac{B^2}{4}+y} - \frac{B}{2}\right) \bmod n$$

在 Rabin 公钥密码系统中，加密算法非常简单，而解密算法却要复杂得多，而且解密算法中出现了一个有限域内求“平方根”的问题。

下面将讨论如何求解模 n 的“平方根”。

因为 $n=pq$，所以可以将模 n 的“平方根”问题：

$$x^2 \equiv c(\bmod n)$$

转化为一个等价的同余方程组：

$$\begin{cases} x^2 \equiv c \pmod p \\ x^2 \equiv c \pmod q \end{cases} \tag{3.11}$$

针对 $x^2 \equiv c \pmod p$，显然要么无解，要么有两个解。

事实上在 Rabin 公钥密码系统中，$x^2 \equiv c \pmod p$ 应该有两个解：

$$x = \pm c^{(p+1)/4}$$

这是因为 $p(\text{mod } 3)=4$，所以 $(p+1)/4$ 是整数，于是：

$$\begin{aligned} &(\pm c^{(p+1)/4})^2 \\ &\equiv c^{(p+1)/2} \pmod p \\ &\equiv c^{(p-1)/2} c \pmod p \\ &\equiv c \pmod p \end{aligned}$$

接下来可将式(3.11)转化为 4 个同余方程组，分别是：

$$\begin{cases} x \equiv c^{(p+1)/4} \pmod p \\ x \equiv c^{(q+1)/4} \pmod q \end{cases} \quad \begin{cases} x \equiv - c^{(p+1)/4} \pmod p \\ x \equiv + c^{(q+1)/4} \pmod q \end{cases}$$

$$\begin{cases} x \equiv + c^{(p+1)/4} \pmod p \\ x \equiv - c^{(q+1)/4} \pmod q \end{cases} \quad \begin{cases} x \equiv - c^{(p+1)/4} \pmod p \\ x \equiv - c^{(q+1)/4} \pmod q \end{cases}$$

以上 4 个方程组可用中国剩余定理来求解。

例 3.22　Rabin 密码系统参数：$p=31, q=71, b=23, n=2201$。

对明文 $x=2003$ 实施 Rabin 加密算法：

$$y = 2003^2 + 23 \times 2003 \bmod 2201 = 1635$$

实施 Rabin 解密算法：

$$x = \sqrt{556+1635} - 1112 \bmod 2201 = \sqrt{2191} - 1112 \bmod 2201$$

接下来求解 $\sqrt{2191} \bmod 2201$，将求解 $\sqrt{2191} \bmod 2201$ 转化为求解方程组：

$$\begin{cases} t = \sqrt{2191} \bmod 31 \\ t = \sqrt{2191} \bmod 71 \end{cases}$$

进一步转化为：

$$\begin{cases} t = \pm 16 \bmod 31 \\ t = \pm 9 \bmod 71 \end{cases}$$

以 $\begin{cases} t=16 \bmod 31 \\ t=9 \bmod 71 \end{cases}$ 的求解为例：

$$\begin{cases} a_1 = 16 \quad m_1 = 31 \quad y_1 = 71^{-1} \bmod 31 = 7 \quad a_1 m_2 y_1 = 1349 \\ a_2 = 9 \quad m_2 = 71 \quad y_2 = 31^{-1} \bmod 71 = 55 \quad a_2 m_1 y_2 = 2139 \end{cases}$$

$$t = (1349 + 2139) \bmod 2201 = 1287$$

$$x_1 = (1287 - 1112) \bmod 2201 = 175$$

同理，解另外 3 个方程组得：

$$x_2 = (790 - 1112) \bmod 2201 = 1879$$

$$x_3 = (1411 - 1112) \bmod 2201 = 299$$
$$x_4 = (914 - 1112) \bmod 2201 = 2003$$

其中，x_4 正是明文。

3.6 习　　题

1. 设 p 和 q 是素数，求 $\varphi(p^m q^n)$。
2. 设 G 是群，试证：对于任意 $a,b \in G$，$o(ab)=o(ba)$。
3. 试证明：半群 $(G,*)$ 是群的充要条件是：对任何 $a,b \in G$，方程组：

$$\begin{cases} ax = b \\ ya = b \end{cases}$$

在 G 中均有解。

4. 令：

$$S = \left\{ \begin{bmatrix} a & b \\ c & d \end{bmatrix} \middle| a,b,c,d \in Z \right\}$$

证明 S 关于矩阵的乘法构成一个半群。

5. “七数剩一，八数剩二，九数剩三，问本数”。（杨辉 1275 年）
6. 试找出乘群 $(Z_{12}^*, *)$ 的全部子群。
7. 试找出加群 $(Z_{12}, +)$ 的全部子群。
8. 设 p 为大于 2 的素数，证明：

$$\left[\left(\frac{p-1}{2}\right)!\right]^2 \equiv (-1)^{\frac{p+1}{2}} (\bmod\ p)$$

9. 设 G 为非空集合，“$*$”为 G 中的双向运算。如果：① $*$ 有结合律；②G 中存在左么元 e'，即对任意的 $a \in G$，总有 $e'a=a$；③G 中任意元素 a 有左逆元 a'，即 $a'a=e'$，则 G 是群。
10. 已知 $p=1999$，试判断 $x=202$ 是否为二次剩余。
11. 计算勒让德符号 $L(1998,2011)$。
12. 求解同余方程组：

$$\begin{cases} x \equiv 3 (\bmod\ 5) \\ x \equiv 6 (\bmod\ 11) \\ x \equiv 8 (\bmod\ 13) \end{cases}$$

13. 求解同余方程：

$$13x \equiv 4 \bmod 99$$

14. 求解二次同余方程：

$$x^2 + 4x + 3 \equiv 0 \bmod 77$$

15. 使用 RSA 公钥密码算法对信息 $m=2004$ 进行加密，RSA 公钥体制的参数是：$p=151$，$q=191$，$n=28\,841$，$a=22\,213$，$b=1777$。
16. 使用 RSA 公钥密码算法对密文 $c=4278$ 进行解密，RSA 公钥体制的参数是：$p=113$，$q=233$，$n=26\,329$，$a=24\,565$，$b=1117$。

17. 分两次使用RSA公钥密码算法对同一个明文进行加密,两个RSA公钥体制的公钥分别是 $n_1=n_2=6077$,$b_1=277$,$b_2=557$,密文分别是 $c_1=2595$,$c_2=3244$,在不分解6077的前提下,求取明文。

18. 使用RSA公钥密码算法对密文 $c=37\ 838$ 进行解密,RSA公钥体制的参数是 $b=37$,$n=50\ 419$,试用循环攻击法求取明文。

19. $n=pq$,p 和 q 都是素数,试用Dixon算法对 n 进行因式分解:

(1) $n=128\ 081$

(2) $n=899\ 777$

(3) $n=658\ 921\ 807$

20. $n=pq$,p 和 q 都是素数,试用Dixon算法对 n 进行因式分解:

(1) $n=142\ 349$

(2) $n=2\ 113\ 721$

(3) $n=62\ 157\ 349$

21. 求解方程:

$$x^2 \equiv 1268 \pmod{1541}$$

22. 求解 $\sqrt{1000} \bmod 2077$。

23. 使用Rabin公钥密码算法对信息 $m=2004$ 进行加密,Rabin公钥体制的参数是:$p=83$,$q=131$,$n=10\ 873$,$b=168$。

24. 使用Rabin公钥密码算法对密文 $c=8008$ 进行解密,Rabin公钥体制的参数是:$p=67$,$q=151$,$n=10\ 117$,$b=199$。

第 4 章　基于离散对数问题的公钥密码系统

与 RSA 和 Rabin 公钥密码系统不同，本章将要介绍的公钥密码系统基于有限域上另外一个 NP 问题：离散对数问题。

离散对数问题又可以分为一般的离散对数问题和椭圆曲线上的离散对数问题。ElGamal 公钥密码系统和椭圆曲线公钥密码系统分别基于这两类问题。

4.1　ElGamal 公钥密码系统

离散对数问题（Discrete Logarithm Problem，DLP）：有限群 $G(Z_p^*)$ 上定义的运算"$*$"，生成元 $\alpha \in G$，$H=\{\alpha^i : i>0\}$ 是由 α 产生的子群，$\beta \in H$，其中 α^i 表示：

$$\underbrace{\alpha * \alpha * \cdots * \alpha}_{i\text{个}}$$

离散对数问题的目标是寻找满足唯一的指数 $0 \leqslant x \leqslant |H|-1$，使得 $\beta=\alpha^x$，也相当于求取对数 $\log_\alpha \beta$。

离散对数问题是难解的，目前还没有找到离散对数问题的多项式时间算法。当然，即使没有找到离散对数问题的多项式时间的求解算法，离散对数问题也是可解的，至少可以用穷举搜索的方法来解决，求解离散对数问题的几种有效的方法将在本章后面的部分中介绍。

T. ElGamal 于 1985 年率先提出 ElGamal 公钥密码系统，ElGamal 系统正是基于离散对数问题的一种公钥密码系统。

ElGamal 公钥密码系统：设 p 是使 Z_p^* 存在难解离散对数问题的素数，$\alpha \in Z_p^*$ 是生成元，则：

（1）密钥生成：

明文空间：　$P=Z_p^*$

密文空间：　$C=Z_p^* \times Z_p^*$

密钥：　$K=\{(p,\alpha,a,\beta): \beta=\alpha^a \pmod p\}$

其中，p、α、β 是公钥；a 是私钥。

（2）加密算法：

对明文 $x \in Z_p^*$，那么密文可以表示为 $(y_1, y_2) \in Z_p^* \times Z_p^*$，且：

$$e_K(x,k) = (y_1, y_2)$$

$$y_1 = \alpha^k \bmod p$$

$$y_2 = x\beta^k \bmod p$$

其中，k 是满足 $k \in Z_{p-1}$ 的随机数。

由于 k 是随机选取的，对应于同一明文可能有多个密文。

(3) 解密算法：

$$d_K(y_1, y_2) = y_2(y_1^a)^{-1} \bmod p$$

其中，y_1、$y_2 \in Z_p^*$。

在 Z_p^* 中，ElGamal 在加密时对明文 x 用 $x\beta^k$ 进行了伪装，同样 α^k 也传给了解密算法。然而，在解密时用 a 得到了 $(y_1^a)^{-1}=(\alpha^{ks})^{-1}=(\beta^k)^{-1}$，因此，能够抵消掉 $y_2=x\beta^k \bmod p$ 中的伪装 β^k，从而得到 x。

例 4.1 在 ElGamal 公钥密码系统中，已知 $p=2111$，明文空间是 Z_{2111}^*，密文空间是 $Z_{2111}^* \times Z_{2111}^*$，设密钥：

$$\alpha = 2,\quad a = 666,\quad \beta = 2^{666} \bmod 2111 = 219$$

对明文 $x=168$ 实施加密(选取参数 $k=555$)：

$$y_1 = 2^{555} (\bmod 2111) = 76$$

$$y_2 = 168 \times 219^{555} (\bmod 2111) = 61$$

实施解密：

$$x = 61 \times (76^{666})^{-1} (\bmod 2111) = 168$$

在 ElGamal 密码体系中，从公开的 α 和 β 求保密的解密密钥就是一个离散对数问题，只要选取合适的素数 p，离散对数问题是难解的。

与 RSA 和 Rabin 公钥密码体制不同，ElGamal 公钥密码体制引入了一个辅助参数 k。k 的存在增加了 ElGamal 的安全性，即使明文和密钥都相同，密文也会随着 k 的不同取值而各不相同，这种非一一对应的特性将增加破解的难度。

4.2 椭圆曲线密码体制

由于原理简单且容易实现，RSA 公钥密码系统一度是应用最广泛的公钥密码系统。但是随着网络和计算机速度的提升，安全强度的需求要求 RSA 算法的密钥长度不断增加。目前学者们普遍认为需要 1024 位以上的密钥才能保障数据的安全性，NIST 更是推荐使用 3072 位的 RSA 密钥。密钥长度的增加大大降低了加解密的速度，同时也显著增加了存储所需的空间和硬件实现的复杂性。1985 年，Neal Koblitz 和 V. S. Miller 基于 ElGamal 公钥密码系统提出了能在低要求的计算环境中达到高强度加密的新型公钥密码系统——椭圆曲线密码体制(Elliptical Curve Cryptography，ECC)。

椭圆曲线：定义在域 F 上的椭圆曲线 E 指的是由韦尔斯特拉斯(Weierstrass)方程

$$y^2z + a_1xyz + a_3yz^2 = x^3 + a_2x^2z + a_4xz^2 + a_6z^3$$

所确定的平面曲线。

满足上述方程的点 (x, y) 称为域 F 上椭圆曲线上的点，除了上面的点，还有一个特殊点——无穷远点 O。

取 $z=1$，可得椭圆曲线的一般方程：

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

有限域 GF(p)上的椭圆曲线：设 p 是素数，且 $p>3$，在有限域 GF(p)上的椭圆曲线由满足：

$$y^2 = x^3 + ax + b(\bmod p) \tag{4.1}$$

的点(x,y)和一个特殊的无穷远点 O 构成，记作 $E_p(a,b)$。其中，a、$b\in GF(p)$且 $4a^3+27b^2\neq 0(\bmod\ p)$。

例 4.2　求椭圆曲线 $y^2=x^3+8x+10(\bmod\ 19)$上的点。

要得到满足椭圆曲线方程的解(x,y)，可利用前面章节中平方剩余的概念。

令 $z=x^3+8x+10(\bmod\ 19)$，先逐个求出 $x=0、1、\cdots、18$ 对应的 z 值，如表 4.1 所示的第 2 列；然后验证 z 是否为模 19 有限域的平方剩余，是否为平方剩余可采用勒让德符号来判定；如果是平方剩余，再计算出模 19 有限域下 z 的平方根。

表 4.1　获得有限域中椭圆曲线上的点

x	z	是否是平方剩余	y	x	z	是否是平方剩余	y
0	10	否		10	7	是	11,8
1	0	0		11	4	是	17,2
2	15	否		12	10	否	
3	4	是	17,2	13	12	否	
4	11	是	7,12	14	16	是	4,15
5	4	是	17,2	15	9	是	16,3
6	8	否		16	16	是	4,15
7	10	否		17	5	是	9,10
8	16	是	4,15	18	1	是	1,18
9	13	否					

根据表 4.1 可得到椭圆曲线 $y^2=x^3+8x+10(\bmod\ 19)$上的点(图 4.1)：

{1,0},{3,2},{3,17},{4,7},{4,12}

{5,2},{5,17},{8,4},{8,15},{10,8}

{10,11},{11,2},{11,17},{14,4},{14,15}

{15,3},{15,16},{16,4},{16,15},{17,9}

{17,10},{18,1},{18,18}

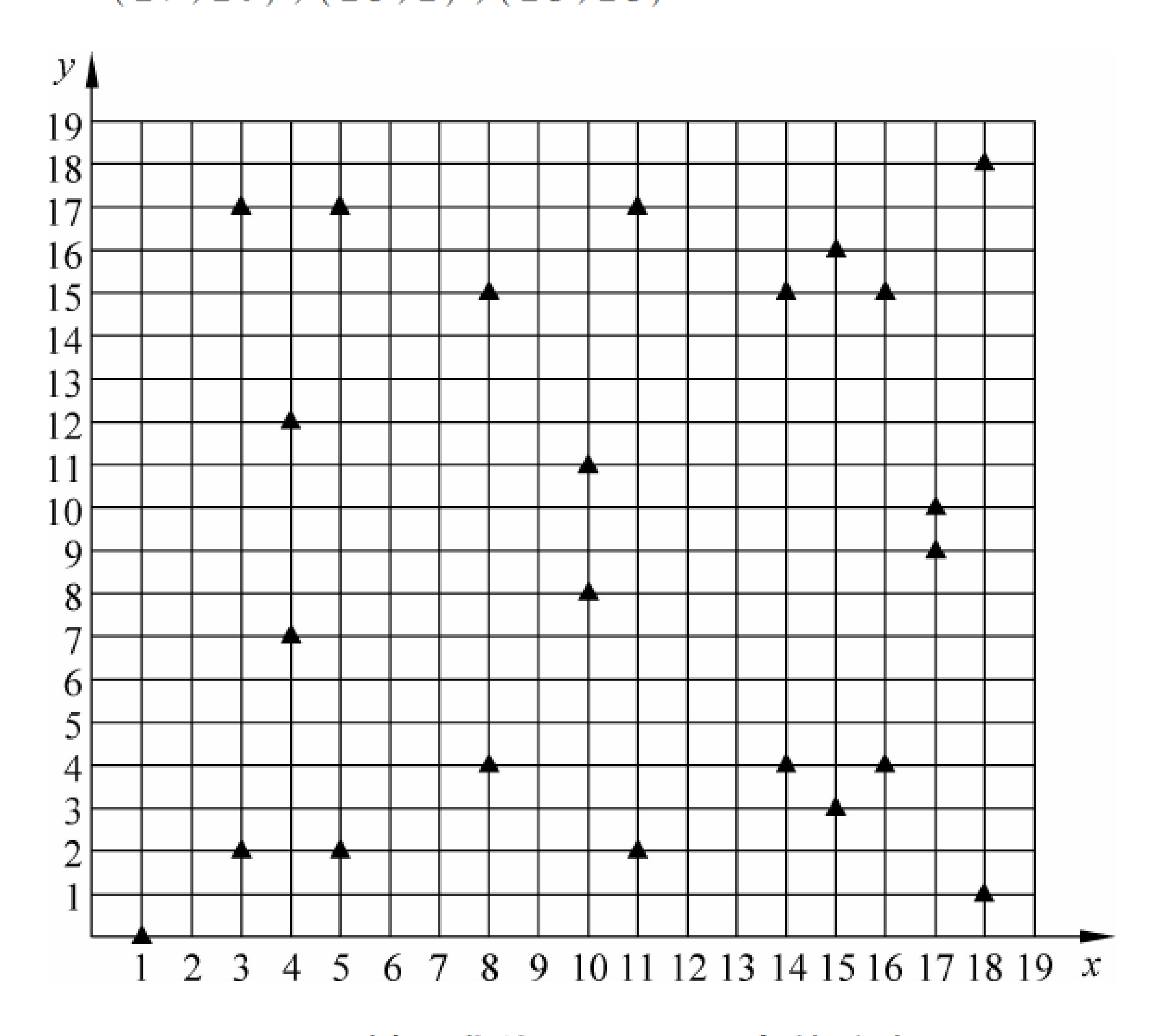


图 4.1　椭圆曲线 $E_{11}(8,10)$点的分布图

求得了椭圆曲线上的点，接下来介绍素数域上椭圆曲线的加法运算。

素数域上椭圆曲线的加法运算，从几何的角度来理解更直观形象。设 P 和 Q 是椭圆曲线上的两点，连接 P 和 Q 两点的直线交椭圆曲线于 R'点，通过 R'点作 y 轴的平行线与椭圆曲线相交于 R，R 点就是 P 和 Q 的和，即 $R=P+Q$，如图 4.2 所示。

特殊情况下，当 P 和 Q 重合时，过 P 点作椭圆曲线的切线，和椭圆曲线相交于 R'。然后通过 R'点作 y 轴的平行线和椭圆曲线相交 R，R 点就是 P 和 P 的和，即 $R=P+P$，此时称 P 和 P 的和运算为椭圆曲线的倍点运算，如图 4.3 所示。

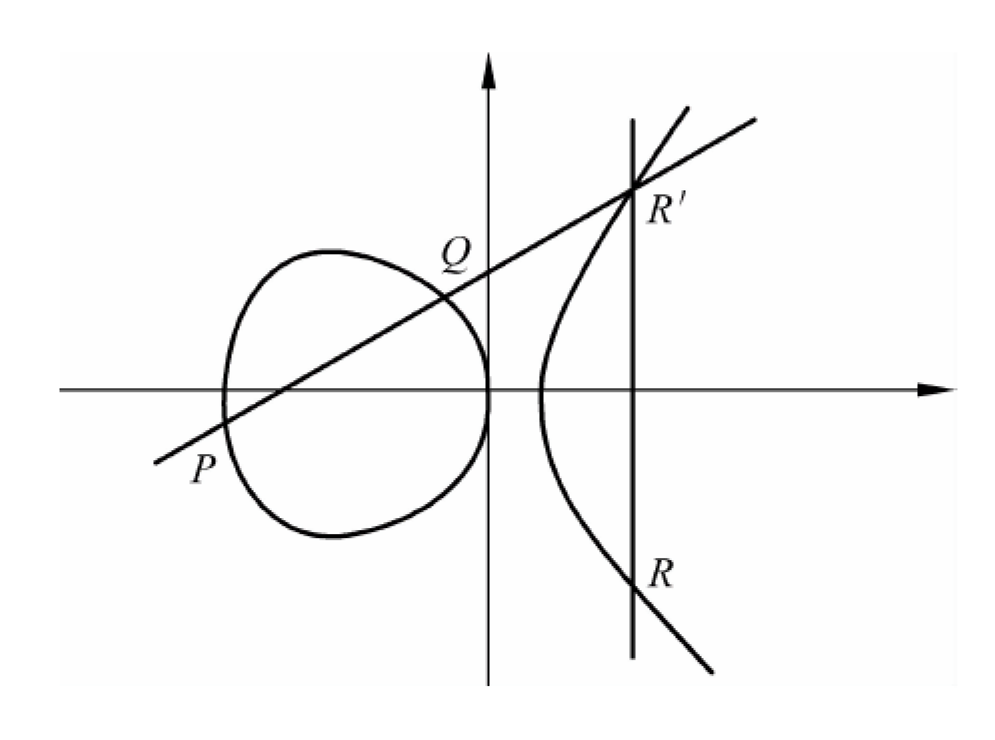


图 4.2 椭圆曲线加法(不同点)

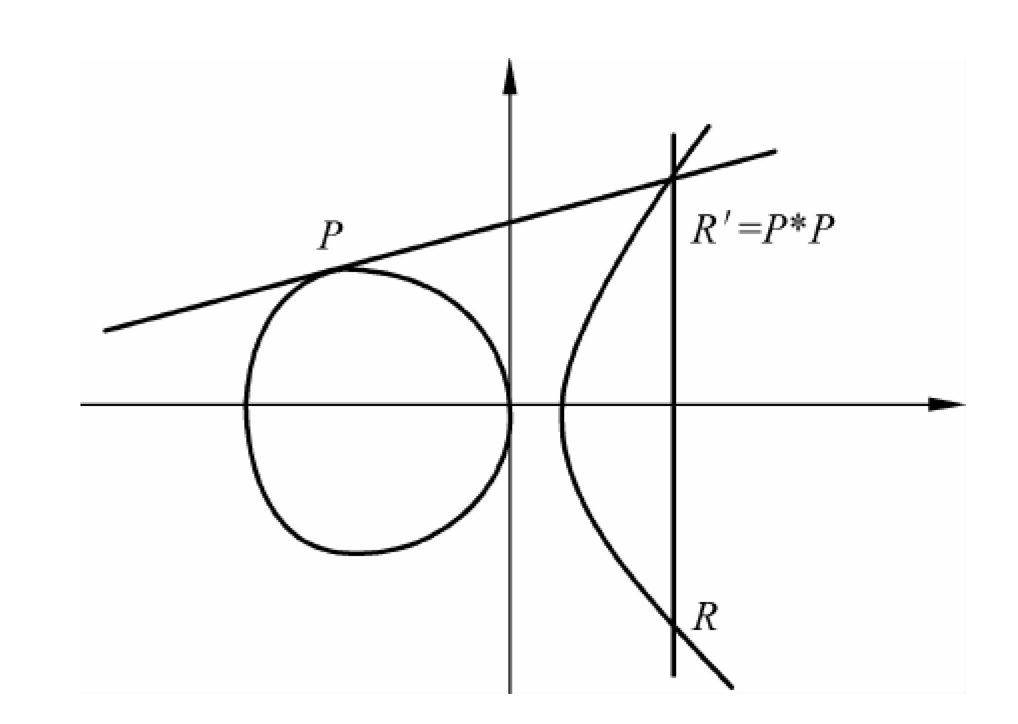


图 4.3 椭圆曲线加法(相同点)

为便于计算，下面给出椭圆曲线上的具体加法代数公式。

设 E 是定义在式(4.1)上的椭圆曲线，$P(x_1,y_1)$、$Q(x_2,y_2)$、$R(x_3,y_3)$是椭圆曲线上的点，且 $R=P+Q$，则有：

当 P 和 Q 不是同一点时，即 $P\neq Q$ 时：

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = -\lambda(x_3 + y_1) \end{cases}$$

其中，$\lambda=\dfrac{y_2-y_1}{x_2-x_1}$(即 PQ 直线的斜率)。

当 P 和 Q 是同一个点时，即 $P=Q$ 时：

$$\begin{cases} x_3 = \lambda^2 - 2x_1 \\ y_3 = -\lambda(x_3 + y_1) \end{cases}$$

其中，$\lambda=\dfrac{dy}{dx}=\dfrac{3x_1^2+a}{2y_1}$(即 PQ 直线的斜率)。

特别地，如果椭圆曲线上两个点关于 x 轴对称，则它们的加法结果定义为无穷远点 O，即：

$$(x_1, -y_1) + (x_1, -y_1) = O$$

点 P 的加法逆元也就是点 P 关于 x 轴的对称点，记为 $-P$，$-P$ 的坐标是$(x_1,-y_1)$。

由此也可以得到一个推论：如果椭圆曲线上处于同一条直线上的 3 个点 A、B、C，那么它们的和 $A+B+C$ 等于无穷远点，即：

$$A + B + C = O$$

例 4.3 椭圆曲线 E:$y^2=x^3+8x+10$，$p=19$，在椭圆曲线上分别取 3 个点 P、Q 和 T：

$$P=(11,17) \quad Q=(15,16) \quad T=(4,7)$$

分别计算 $M=P+Q$,$N=Q+T$,$U=M+T$,$V=P+N$。

(1) 计算 M:

$$\lambda_M=\frac{16-17}{15-11} \bmod 19=14$$
$$y_{M0}=(17-14\times 11) \bmod 19=4$$
$$x_M=(14^2-11-15) \bmod 19=18$$
$$y_M=[-18\times(11-8)-17] \bmod 19=18$$

即 M 的坐标是(18,18)。

(2) 计算 N:

$$\lambda_N=\frac{7-16}{4-15} \bmod 19=6$$
$$x_N=(6^2-15-4) \bmod 19=17$$
$$y_N=[6\times(15-17)-16] \bmod 19=10$$

即 N 的坐标是(17,10)。

(3) 计算 U:

$$\lambda_U=\frac{7-18}{4-18} \bmod 19=13$$
$$x_U=(13^2-18-4) \bmod 19=14$$
$$y_U=[13\times(18-14)-18] \bmod 19=15$$

即 U 的坐标是(14,15)。

(4) 计算 V:

$$\lambda_V=\frac{10-17}{17-11} \bmod 19=2$$
$$x_V=(22-11\times 17) \bmod 19=14$$
$$y_V=[2\times(11\times 14)\times 17] \bmod 19=15$$

即 N 的坐标是(14,15)。

计算得到 $U=V$,即$(P+Q)+T=P+(Q+T)$,验证了素数域中椭圆曲线上的点对运算“+”满足结合律。

例 4.4　椭圆曲线 E:$y^2=x^3+8x+10$,$p=19$,在椭圆曲线上分别取点 P 和 Q:

$$P=(3,2), \quad Q=(5,17)$$

分别计算 $4Q$ 和 $6P$。

先计算 $2Q$:

$$\lambda=\frac{3\times 5^2+8}{2\times 17} \bmod 19=3$$
$$x=(3^2-5-5) \bmod 19=18$$
$$y=[3\times(5-18)-17] \bmod 19=1$$

$2Q$ 的坐标是(18,1)。

然后计算 $4Q=2Q+2Q$:

$$\lambda=\frac{3\times 18^2+8}{2\times 1} \bmod 19=15$$

$$x = (15^2 - 18 - 18) \bmod 19 = 18$$

$$y = [18 \times (18 - 18) - 1] \bmod 19 = 18$$

$4Q$ 的坐标是(18,18),同理,可算出 $6P=(8,15)$。

MV-椭圆曲线公钥密码系统:设椭圆曲线 E 定义在 $Z_p(p>3)$上,E 包含一个循环子群 H 使得离散对数问题难解。

明文空间:$P=Z_p \times Z_p$

密文空间:$C=E \times Z_p \times Z_p$

密钥:$K=\{(E,\alpha,\beta,a):\beta=a\alpha=\alpha+\cdots+\alpha(共\ a\ 个)\}$,其中,$\alpha \in E$,$\alpha$、$\beta$ 是公钥,a 是私钥。

(1) 加密算法:

$$e_K(x,t) = (y_0, y_1, y_2)$$

其中,t 是满足 $t \in Z_{|H|}$ 的任选参数,加密过程如下:

$$y_0 = t\alpha$$

$$(c_1, c_2) = t\beta$$

$$y_1 = c_1 x_1 \bmod p$$

$$y_2 = c_2 x_2 \bmod p$$

(2) 解密算法:

$$d_K(y) = (y_1 c_1^{-1} \bmod p, y_2 c_2^{-1} \bmod p)$$

其中,$(c_1,c_2)=a y_0$。

例 4.5 定义在有限域上的椭圆曲线 E: $y^2=x^3+8x+10$, $p=23$,明文空间由 484(22×22=484)个明文构成,取明文 $x=(19,13)$,求其加解密过程。

选择椭圆曲线上的点:

$$\alpha = (7,8)$$

令 $a=17$,则:

$$\beta = 17 \times (7,8) = (10,20)$$

选取 $t=3$,则:

$$y_0 = 3 \times (7,8) = (22,22)$$

$$(c_1, c_2) = 3 \times (10,20) = (18,12)$$

$$y_1 = 19 \times 18 \bmod 23 = 20$$

$$y_2 = 13 \times 12 \bmod 23 = 18$$

$(y_0,y_1,y_2)=((22,22),20,18)$就是加密的结果。

实施解密:

$$(c_1, c_2) = 17 \times (22,22) = (18,12)$$

$$x_1 = 20 \times 18^{-1} \bmod 23 = 19$$

$$x_2 = 18 \times 12^{-1} \bmod 23 = 13$$

在相同的安全性下,ECC 所需要的密钥量比 RSA 少,160 位的 ECC 可以提供与 1024 位的 RSA 或 DSA 相当的安全强度。椭圆曲线的运算也很容易在计算机的软件和硬件上完成。对于带宽和存储受限的环境,特别是在移动通信领域,ECC 是目前最理想的公钥算法之一。

虽然 ECC 效率很高，但 ECC 的应用面临一些实施中的问题，如椭圆曲线标量乘法问题、椭圆曲线阶的计算问题和基点选取问题等。

4.3　椭圆曲线标量乘法

在 ECC 的实现中最耗时的运算就是椭圆曲线标量乘法，即 kP 的计算。下面介绍几种常用的标量乘法算法。

4.3.1　二进制法

二进制表示法是 ECC 标量乘法中标量 k 的最基本表示方法，即：

$$k = \sum_{i=0}^{l-1} k_i 2^i$$

其中，$k_i \in \{0,1\}$，$l = \lfloor \log_2 k \rfloor + 1$。

二进制表示法分为从右向左和从左向右两种。

(1) 从右向左的二进制算法描述如下：

① $Q=0$，$l=\lfloor \log_2 k \rfloor +1$。

② 对于 i 从 0 到 $l-1$，重复执行。

若 $k_i=1$，则 $Q=Q+P$；

否则 $P=2P$。

③ 返回(Q)。

例 4.6　椭圆曲线 $E_{211}(1,1)$：$y^2=x^3+x+1$，$p=211$，已知曲线上的点是 $P(2,86)$，$k=112$，求 $Q=kP$。

先进行初始化：

$$Q = 0;\ k = (1110000)_2$$

从右向左计算过程如下：

$$0:\ P = 2P = (43,209)$$
$$0:\ P = 2P = (207,12)$$
$$0:\ P = 2P = (14,4)$$
$$0:\ P = 2P = (55,116)$$
$$1:\ Q = Q + P = (55,116);P = 2P = (157,201)$$
$$1:\ Q = Q + P = (29,187);P = 2P = (151,65)$$
$$1:\ Q = Q + P = (32,27)$$

返回结果：$Q=(32,27)$。

(2) 从左向右的二进制算法可描述如下：

① $Q=0$ ，$l=\lfloor \log_2 k \rfloor +1$。

② 对于 i 从 0 到 $l-1$，重复执行：

$Q=2Q$

若 $k_i=1$,则 $Q=Q+P$。

③ 返回(Q)。

例 4.7　椭圆曲线 $E_{211}(1,1)$：$y^2=x^3+x+1$,$p=211$,已知曲线上的点是 $P(2,86)$,$k=112$,求 $Q=kP$。

先进行初始化：

$$Q=0;\ k=(1110000)_2$$

从左向右计算过程如下：

$$1:\ Q=2Q=0;\ Q=Q+P=(2,86)$$
$$1:\ Q=2Q=(43,209);\ Q=Q+P=(175,28)$$
$$1:\ Q=2Q=(12,105);\ Q=Q+P=(93,142)$$
$$0:\ Q=2Q=(45,190)$$
$$0:\ Q=2Q=(137,178)$$
$$0:\ Q=2Q=(13,177)$$
$$0:\ Q=2Q=(32,27)$$

返回 $Q=(32,27)$。

4.3.2　带符号二进制法

1951 年 Booth 提出了标量的带符号二进制表示,后来 Rietweisner 证明了对于每一个非负整数 k 均可以唯一地表示为：

$$k=\sum_{i=0}^{l-1}k_i2^i$$

其中,$k_i\in D_s=\{0,\pm 1\}$,$l=\lfloor \log_2 k\rfloor+1$。

(1) 带符号二进制表示法算法如下：

① $i=0$, $l=\lfloor \log_2 k\rfloor+1$。

② 当 $k\geqslant 1$ 时,重复执行：

若 k 是奇数,则 $k_i=2-(k \bmod 4)$,$k=k-k_i$;

否则,$k_i=0$。

③ $k=k/2$,$i=i+1$。

④ 返回$(k_{i-1},k_{i-2},\cdots,k_1,k_0)_2$。

例 4.8　已知 $k=112$,则求带符号二进制表示 k 的过程如下：

$i=0$

$k=112$ 是偶数,　$k_0=0$;　$k=56$,　$i=1$

$k=56$ 是偶数,　$k_1=0$;　$k=28$,　$i=2$

$k=28$ 是偶数,　$k_2=0$;　$k=14$,　$i=3$

$k=14$ 是偶数,　$k_3=0$;　$k=7$,　$i=4$

$k=7$ 是奇数,　$k_4=-1$,　$k=8$;　$k=4$,　$i=5$

$k=4$ 是偶数,　$k_5=0$;　$k=2$,　$i=6$

$k=2$ 是偶数,　$k_6=0$;　$k=1$,　$i=7$

$k=1$ 是奇数,　$k_7=1$,　$k=0$;　$k=0$,　$i=8$

返回$(100-10000)_2$。

(2) 针对带符号二进制表示的情况,相应的标量乘法算法描述如下:

① 用上面的算法计算

$$k = \sum_{i=0}^{l-1} k_i 2^i$$

② $Q=0$。

③ 对于 i 从 $l-1$ 到 0,重复执行:

$Q=2Q$

若 $k_i=1$,则 $Q=Q+P$;

若 $k_i=-1$,则 $Q=Q-P$。

④ 返回 Q。

例 4.9　椭圆曲线 $E_{211}(1,1)$: $y^2=x^3+x+1$, $p=211$,已知曲线上的点是 $P(2,86)$,求 $Q=kP$, $k=112$。

用前面的算法先计算出 $k=(100-10000)_2$,接下来的过程如下:

$$\begin{aligned}
&Q = 0\\
&1:\ Q = 2Q = 0;\ Q = Q + P = (2,86)\\
&0:\ Q = 2Q = (43,209)\\
&0:\ Q = 2Q = (207,12)\\
&-1:\ Q = 2Q = (14,4);\ Q = Q - P = (93,142)\\
&0:\ Q = 2Q = (45,190)\\
&0:\ Q = 2Q = (137,178)\\
&0:\ Q = 2Q = (13,177)\\
&0:\ Q = 2Q = (32,27)
\end{aligned}$$

返回结果 $Q=(32,27)$。

4.3.3　Comb 标量乘算法

对于固定基点的标量乘运算,在存储容量允许的条件下,可以通过预计算并存储一些与基点 P 相关的数据来加快运算的速度,固定基点的 Comb 算法正是基于这样的思路。

设 $(k_{l-1}\cdots k_1 k_0)_2$ 为 k 的二进制表示形式:

$$k = \sum_{i=0}^{l-1} k_i 2^i \quad k \in \{0,1\}$$

w 表示窗口宽度,$w \geqslant 2$,计算 $d=\lceil t/w \rceil$。

若 k 不足 $w \times d$ 位,则在 $l \sim (w \times d-1)$ 位补 0,然后将 $k=(k_{l-1}\cdots k_1 k_0)_2$ 分成 w 段,即将 $k=(k_{l-1}\cdots k_1 k_0)_2$ 写成 $k=K_{w-1} \| \cdots \| k_1 \| k_0$ 的形式,每个 K_j 都是长度为 d 的二进制串,于是可将 k 表示成 w 行 d 列的矩阵形式,即:

$$k = \sum_{i=0}^{w-1} \sum_{j=0}^{d-1} K_j^i 2^{i*d+j}, \quad K_j^i \in \{0,1\}$$

$$k = \begin{bmatrix} K^0 \\ \vdots \\ K^i \\ \vdots \\ K^{w-1} \end{bmatrix} = \begin{bmatrix} K_{d-1}^0 & \cdots & K_0^0 \\ \vdots & & \vdots \\ K_{d-1}^i & \cdots & K_0^i \\ \vdots & & \vdots \\ K_{d-1}^{w-1} & \cdots & K_0^{w-1} \end{bmatrix} = \begin{bmatrix} k_{d-1} & \cdots & k_0 \\ \vdots & & \vdots \\ k_{d(i+1)-1} & \cdots & k_{di} \\ \vdots & & \vdots \\ k_{wd-1} & \cdots & k_{(w-1)d} \end{bmatrix}$$

P 是椭圆曲线上的一个点,对于所有长度为 w 的二进制串$(a_{w-1},\cdots,a_1,a_0)$,有:

$$(a_{w-1},\cdots,a_1,a_0)P = a_{w-1}2^{(w-1)d}P + \cdots + a_1 2^d P + a_0 P$$

Comb 标量乘算法可描述如下:

(1) $d=\lceil t/w \rceil$。

(2) 计算 2^dP、$2^{2d}P$、$\cdots$、$2^{(w-1)d}P$,然后对于所有长度为 w 的二进制串$(a_{w-1},\cdots,a_1,a_0)$,预计算$(a_{w-1},\cdots,a_1,a_0)P$。

(3) 将 k 表示成 w 行 d 列的矩阵形式:

$$Q = [K_{d-1}^{w-1},\cdots,K_{d-1}^{1},K_{d-1}^{0}]P$$

(4) 对于 i 从 $d-1$ 到 0,重复执行:

$$Q = 2Q$$

$$Q = Q + [K_i^{w-1},\cdots,K_i^1,K_i^0]P$$

(5) 返回 Q。

4.4 椭圆曲线的阶和基点

判断选取的椭圆曲线是否是非奇异椭圆曲线,是否是安全曲线,需要计算椭圆曲线的阶。

如果椭圆曲线的一点 P,存在最小的正整数 n,使得数乘 $nP=O$(无穷远点),则将 n 称为 P 的阶。

例 4.10 椭圆曲线 E:$y^2=x^3+8x+10$,$p=19$,在椭圆曲线上分别取点 P 和 Q:

$$P = (3,2) \quad Q = (5,17)$$

通过计算,可以发现 Q、$2Q$、$3Q$、$4Q$、$5Q$、$6Q$、$7Q$、$8Q$、$\cdots$的结果分别是(5,17)、(18,1)、(1,0)、(18,18)、(5,2)、O、(5,17)、(18,1)、$\cdots$,出现周期为 6 的循环,且有 $6Q=O$。

对照群的性质,称 Q 的阶为 6。

同样也可以得到 P 的阶为 24,Q、$2Q$、$\cdots$、$24Q$ 分别是(3,2)、(10,8)、(11,2)、(5,17)、(15,3)、(8,15)、(14,15)、(18,1)、(4,12)、(17,10)、(16,15)、(1,0)、(16,4)、(17,9)、(4,7)、(18,18)、(14,4)、(8,4)、(15,16)、(5,2)、(11,17)、(10,11)、(3,17)、O。

P 和 Q 的性质是不同的:椭圆曲线上的每一个点都可以表示成 P 的指数形式,而并不是所有点都能表示成 Q 的指数形式,能表示成 Q 的指数形式的点算上 O 仅有 6 个。

对照群的性质,P 是椭圆曲线上的生成元。

虽然对于小的参数能够采用穷举法,但当椭圆曲线参数较大时,阶的计算变得比较困难。目前主流的求解算法有 SEA 算法和 Satoh 算法,感兴趣的读者可查阅相关文献。

在确定了椭圆曲线之后,找出椭圆曲线加群或其大素因子子群的一个生成元,即基点,也是相当重要的。

在小参数情况下,基点选取的步骤如下:

(1) 椭圆曲线 $E_p(a,b)$上点的计算。

首先对每个满足 $0\leqslant x<p$ 的 x,计算 $x^3+ax+b \pmod p$;然后对上一步骤得到的每个结果确定它是否有一个模 p 的平方根;如果没有,那么在 $E_p(a,b)$中就没有一个横坐标为

x 的点；如果有，就有两个 y 值(除非 y 值为零)。这些 (x,y) 值就是椭圆曲线 $E_p(a,b)$ 上的点。

(2) 椭圆曲线上点阶的计算。

首先选取椭圆上一点 $q(x_0,y_0)$，然后计算 $q+q+\cdots+q$，使得 $nq=0$ 成立的最小值 n。

(3) 检测 n 是否是素数，如果不是，返回第(2)步。

例 4.11　椭圆曲线 $E_{211}(1,1)$：$y^2=x^3+x+1$，$p=211$，取曲线上的点 $v(2,86)$，计算该点的阶，即计算使 $nv=0$ 的 n 值。

$$
\begin{aligned}
2v &= v+v = (2,86)+(2,86) = (43,209)\\
3v &= 2v+v = (175,28)\\
4v &= 3v+v = (207,12)\\
5v &= 4v+v = (201,62)\\
&\vdots\\
222v &= 221v+v = (2,125)
\end{aligned}
$$

因为 $222v=(2,125)=(2,-86)=-v$，所以 $223v=222v+v=0$(无穷远点)，223 即为点 v 的阶。

经计算得 $n=223$。验证得知 $n=223$ 是一个素数，所以点 v 可以作为基点。

鉴于前面已经介绍了椭圆曲线的标量乘、元素的阶、基点，下面给出一个实际的椭圆曲线公钥系统实例。

例 4.12　在 P 为 256 位的情况下，设定椭圆曲线 $E_p(a,b)$：$y^2=x^3+ax+b$，其中：

$$p = 2^{256}-2^{224}+2^{192}+2^{96}-1$$

$$a = -3 \pmod{p}$$

$b=$0x	5AC635D8	AA3A93E7	B3EBBD55	769886BC
	651D06B0	CC53B0F6	3BCE3C3E	27D2604B

基点 G 取为 (x,y)，其中：

$x=$0x	6B17D1F2	E12C4247	F8BCE6E5	63A440F2
	77037D81	2DEB33A0	F4A13945	D898C296
$y=$0x	4FE342E2	FE1A7F9B	8EE7EB4A	7C0F9E16
	2BCE3357	6B315ECE	CBB64068	37BF51F5

计算基点的阶，可得：

$n=$0x	FFFFFFFF	00000000	FFFFFFFF	FFFFFFFF
	BCE6FAAD	A7179E84	F3B9CAC2	FC632551

在区间 $[1,n-1]$ 中随机选取一个整数 d：

$d=$0x	02BEEAAF	F21EBDD3	44CACFF5	31A49515
	90782720	746EA776	C34D309E	FFAE5AAB

计算点 $Q=dG$(d 个 G 相加)：

$x_Q=$0x	2A5D8E9B	9216DB25	79BB64CB	108C22D4
	268C8DC3	BEC8F94A	8B6FC3DF	A2984D2F
$y_Q=$0x	043D9CFB	E54CC922	DD975CAA	E0F82EBE
	0FE88607	2AB3D064	16A9943C	D1B087D8

接收方公开自己的公开密钥($E(F_q)$,G,n,Q);

发送方要发送消息 m 给接收方,于是将 m 表示成一个域元素 $m\in F_q$,设:

m=0x 2EAD

在区间[1,$n-1$]内选取一个随机数 k:

k=0x FC246BB9 7577653D 5417FEE4 E7C7A495
82C1C1E6 F704C223 6CC9C719 5B866A08

依据 Bob 的公钥计算点(x_1, y_1)=kG:

x_1=0x 85960405 FB27A306 7F4B35B3 316EA176
FD6F7FFA 4E908539 82CD39D0 60A15FD9

y_1=0x 5CDB65D9 E42AC7F9 9FA543CD F6A84BDC
EB74F9F5 23BDC6F1 B4D8087B B2256E60

计算点(x_2, y_2)=kQ:

x_2=0x E2096AC7 01789F63 E88DD15C 69E03E9F
804166A8 9DEE9F59 35719BF6 2F3402C6

y_2=0x 0A6A0174 E9A6D109 59CFC4D4 EEECE23C
A8313141 60BB91AF A432FF3E 64872105

计算 $C=m*x_2$:

C=0x 718D13F5 AB278117 A3751679 D7CAFAD5
6CA4831D 9AE3B6E3 87CCA1D9 3EA59D04

传送加密数据(x_1,y_1,C)给接收方。

接收方收到密文(x_1,y_1,C)后,用私钥 d 计算点(x_2,y_2)=d(x_1,y_1),再计算 F_q 中 x_2^{-1} 的值:

x_2=0x E2096AC7 01789F63 E88DD15C 69E03E9F
804166A8 9DEE9F59 35719BF6 2F3402C6

y_2=0x 0A6A0174 E9A6D109 59CFC4D4 EEECE23C
A8313141 60BB91AF A432FF3E 64872105

x_2^{-1}=0x 38061B20 9B86E8B6 9187F19A 559C7BEB
53AE7D38 4B4D2DC1 5C4C44C4 81C71D08

通过计算 $m=Cx_2^{-1}$,恢复出明文数据 m:

$$m = 0\text{x}\ \ 2\text{EAD}$$

4.5 GF(2^m)域的椭圆曲线

前面介绍了素数域上的椭圆曲线,从应用的角度来考虑,GF(2^m)域的运算比在素数域上的运算更容易用硬件来实现,所以定义在 GF(2^m)上的椭圆曲线也是一个不错的选择。

GF(2^m)上的椭圆曲线由满足:

$$y^2+xy\equiv x^3+ax^2+b \tag{4.2}$$

的点(x,y)和一个特殊的无穷远点 O 构成。

其中，a、$b \in GF(2^m)$且 $b \neq 0$。

阶为 2^m 的有限域又称为二进制域或者是特征是 2 的有限域。

$GF(2^m)$的表示方法主要有两种：多项式基(Polynomial Basis)表示法和正规基(Normal Basis)表示法。多项式基更易于理解，因此本书主要介绍基于多项式基表示法的椭圆曲线。

在多项式基表示法中，$GF(2^m)$域上的元素的最高次数是 $m-1$ 次，表示如下：

$$a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0$$

其中，$a_i \in GF(2^m)$，$i \in \{0,1,2,\cdots,m-1\}$。

例 4.13　二进制域 $GF(2^4)$上的所有元素，如表 4.2 所示。

表 4.2　多项式基表示法

0	x^2	x^3	x^3+x^2
1	x^2+1	x^3+1	x^3+x^2+1
x	x^2+x	x^3+x	x^3+x^2+x
$x+1$	x^2+x+1	x^3+x+1	x^3+x^2+x+1

取 $GF(2^4)$域上的不可约多项式是 $f(x)=x^4+x+1$，定义 $GF(2^4)$域上的运算：

加法：

$$(x^2+1)+(x^3+x^2+x) = x^3+x+1$$

乘法：

$$\begin{aligned} & (x^3+x^2+1) * (x^2+x+1) \\ = & (x^5+x^4+x^3)+(x^4+x^3+x^2)+(x^2+x+1) \\ = & (x^5+x+1) \bmod (x^4+x+1) \\ = & x^2+1 \end{aligned}$$

减法运算与加法运算相同，如：

减法：

$$(x^3+x^2+x+1)-(x^2+x) = x^3+1$$

乘法逆运算由乘法得到，如：

求逆：$(x^3+x^2+1)^{-1}=x^2$，因为$(x^3+x+1) * (x^2)=1 \bmod (x^4+x+1)$

不可约多项式是指不能分解的多项式。表 4.3 给出了 NIST 推荐的不可约多项式。

表 4.3　NIST 推荐的不可约多项式

有限域	不可约多项式	有限域	不可约多项式
$GF(2^{163})$	$f(x)=x^{163}+x^7+x^6+x^3+1$	$GF(2^{409})$	$f(x)=x^{409}+x^{87}+1$
$GF(2^{233})$	$f(x)=x^{233}+x^{74}+1$	$GF(2^{571})$	$f(x)=x^{571}+x^{10}+x^5+x^2+1$
$GF(2^{283})$	$f(x)=x^{283}+x^{12}+x^7+x^5+1$		

基于 $GF(2^m)$域上的运算可以得到 $GF(2^m)$域上椭圆曲线的运算。

设 E 是定义在式(4.2)上的椭圆曲线，$P(x_1,y_1)$、$Q(x_2,y_2)$、$R(x_3,y_3)$是椭圆曲线上的点，且 $R=P+Q$，$GF(2^m)$域上椭圆曲线的运算如下：

(1) $O+P=P+O=P$。

(2) P 点的逆：$-P=(x_1,x_1+y_1)$。

(3) 当 P 和 Q 不是同一点时，即 $P\neq Q$ 时：

$$\begin{cases} x_3 = \lambda^2+\lambda+x_1+x_2+a \\ y_3 = \lambda(x_1+x_2)+x_3+y_1 \end{cases}$$

其中，$\lambda=\dfrac{y_1+y_2}{x_1+x_2}$。

(4) 当 P 和 Q 是同一个点时，即 $P=Q$ 时：

$$\begin{cases} x_3 = \lambda^2+\lambda+a \\ y_3 = x_1^2+(\lambda+1)x_3 \end{cases}$$

其中，$\lambda=x_1+\dfrac{y_1}{x_1}$(即 PQ 直线的斜率)。

例 4.14 $f(z)=z^4+z+1$ 是域 $GF(2^4)$ 上的不可约多项式，椭圆曲线 E：$y^2+xy=x^3+z^3x^2+(z^3+1)$，曲线上除无穷远点 O 的所有点是：

(0000,1011)	(0001,0000)	(0001,0001)
(0010,1101)	(0111,1100)	(0011,1100)
(0011,1111)	(0101,0000)	(0101,0101)
(0111,1011)	(1011,1001)	(1000,0001)
(1000,1001)	(1001,0110)	(1001,1111)
(1011,0010)	(1100,0000)	(1100,1100)
(1111,0100)	(1111,1011)	(0010,1111)

点加运算：

$$(0010,1111)+(1100,1100)=(0001,0001)$$

标量乘运算：

$$2(0010,1111)=(1011,0010)$$

4.6 离散对数问题的求解

ElGamal 公钥密码算法基于离散对数问题(Discrete Logarithm Problem，DLP)，而椭圆曲线公钥密码算法基于椭圆曲线上的离散对数问题(Elliptic Curve Discrete Logarithm Problem，ECDLP)。

离散对数问题前面已经描述过，椭圆曲线上的离散对数问题与一般的离散对数问题类似，具体描述是给定一条有限域上的椭圆曲线以及椭圆曲线的两个点 P 和 Q，寻找一个整数 k，使得 P$=k$Q，如果这样的数存在，这就是椭圆曲线离散对数；即选取椭圆曲线上的一个点 P(x,y)，作为一个基点，那么给定一个整数 k，计算 Q$=k$P 是容易的，但是已知了点 P 和 Q 想要求出 k 则是非常困难的。

目前，对 DLP 和 ECDLP 的攻击一般有以下一些方法。

1. 穷尽搜索法

穷尽搜索法是求解离散对数最简单的方法，通过依次计算 α^0、α^1、α^3、…、α^n，直到得到使

得 $\alpha^x=\beta$ 的 x 的值，这个方法需要的时间复杂度是 $O(n)$，空间复杂度是 $O(1)$，n 就是 α 的阶，因此当 n 很大时，这个方法是不可行的。

2. 大步小步法

大步小步法(Baby Step/Giant Step)是由 Shanks 提出的，在 1978 年以前该算法是求解 ECDLP 的最好方法。

假设群的阶是 n，点 α 的阶也是 n，设 $\beta=\alpha^x$，令 $L=\lfloor\sqrt{n}\rfloor$，那么 x 可以表示成 $x=iL+j$ 的形式，其中，$0\leqslant i、j<L$，因此 $\alpha^x=\alpha^{i*L}\alpha^j$，可以得到 $\beta(\alpha^L)^{-i}=\alpha^j$，由此可得 x 的计算方法。

设 $L=\lfloor\sqrt{n}\rfloor$，建立一个表 $A(\alpha^j,j)$，这里 $0\leqslant j<L$，表的顺序按 α^j 进行排序，其中 j 表示的是"小步"。

依次计算 $\beta(\alpha^L)^{-i}$，其中 $0\leqslant i<L$，表示的是"大步"。然后在上面的表 A 中进行查询，如果 α^j 中存在和 $\beta(\alpha^L)^{-i}$ 相同的值，则 $x=iL+j$。

例 4.15　令 $p=181$，$\alpha=23$ 是阶为 180 的生成元，$\beta=113$，求离散对数 $\log_3 113$ 的值。

解：

$$L=\lfloor\sqrt{180}\rfloor=14$$

$$\alpha^{-1}=65,\quad \alpha^{-L}=65^{14}(\mathrm{mod}\ 181)=20$$

建立一个表 $A(j,\alpha^j)$，$0\leqslant j<14$，以 α^j 值的大小进行排序，得到结果如表 4.4 所示。

表 4.4　α^j 的计算结果表

j	0	4	1	3	8	7	9	10	12	11	6	13	5	2
$23^j(\mathrm{mod})181$	1	15	23	40	44	57	107	108	117	131	152	157	164	167

对 i 分别取 0、1、2、…、13，依次计算 $113*23^{-14*i}$ 的值，如表 4.5 所示。

表 4.5　$113*23^{-14*i}$ 的计算结果表

i	0	1	2	3
$113*23^{-14*i}(\mathrm{mod}\ 181)$	113	88	131	40

$i=2$ 时，$113*23^{-14*i}(\mathrm{mod}\ 181)=131$ 在表 4.4 中就出现了 $j=11$。

因为 $x=i*L+j=2*14+11=39$，所以 $\log_3 113=39$。

大步小步法需要存储 $O(\sqrt{n})$ 个元素，构造表需要 $O(\sqrt{n})$ 个乘法和 $O(\sqrt{n}\log n)$ 次比较。在大步小步法中需要 $O(\sqrt{n})$ 个乘法，每次乘法的结果需做 $O(\log n)c$ 次比较，所以该算法的时间复杂度和空间复杂度都是 $O(\sqrt{n}\log n)$。大步小步法在时间复杂度上的减小是以空间复杂度的增加为代价的。

3. Pollard ρ 方法

1978 年 Pollard 提出了一种以概率求解的方法，其时间复杂度与大步小步法相当，约为 $O\sqrt{\pi n/2}$，但是其空间复杂度只有 $O(1)$，在这一点上该算法优于大步小步法。

算法可描述如下：首先将群 G 分成 3 个大小相当的子集 S_1、S_2 和 S_3，然后随机的选取 a_0 和 b_0，计算 $x_0=\alpha^{a_0}\beta^{b_0}$，按照以下的方式进行定义序列：

$$x_{i+1}=\begin{cases}\beta x_i, x_i \in S_1\\ x_i^2, x_i \in S_2\\ \alpha x_i, x_i \in S_3\end{cases}$$

$$a_{i+1}=\begin{cases}a_i, x_i \in S_1\\ 2a_i, x_i \in S_2\\ a_i+1, x_i \in S_3\end{cases}$$

$$b_{i+1}=\begin{cases}b_i+1, x_i \in S_1\\ 2b_i, x_i \in S_2\\ b_i, x_i \in S_3\end{cases}$$

$\{x_i\}$组成的子集满足 $x_i=\alpha^{a_i}\beta^{b_i}$，其中 i 大于等于 0，由于$\{x_i\}$会重复，设正整数 i 不等于 j，满足 $x_i=x_j$，设 $l=\log_\alpha\beta$ 则：

$$\alpha^{a_i+lb_i}=\alpha^{a_i}\beta^{b_i}=x_i=x_j=\alpha^{a_j+lb_j}=\alpha^{a_j}\beta^{b_j}$$

所以 $x(b_j-b_i)=a_i-b_j \pmod n$。

若 $\gcd(n,b_j-b_i)=1$，则 x 的值只有一个，$x=((b_j-b_i)\pmod n)^{-1}*(a_i-b_j)\pmod n$。

例 4.16 设 $p=179$ 的群 G 中，$\alpha=3$，$\beta=75$，α 阶 n 为 89，求 $x=\log_\alpha\beta$。

解：首先将群 G 分为规模相当的 3 个集合 S_1、S_2、S_3：

$$S_1=[1,59],\quad S_2=[60,118],\quad S_3=[119,178]$$

随机选取 a_0、b_0：

$$a_0=133,\quad b_0=158$$

计算 $x_0=\alpha^{a_0}\beta^{b_0}=57 \bmod (p)$，并根据上面的规则计算出 x_i、a_i、b_i 值，如表 4.6 所示。

表 4.6 x_i、a_i、b_i 的值

i	x_i	a_i	b_i	i	x_i	a_i	b_i
0	25	133	158	15	89	34 049	40 810
1	85	133	159	16	45	68 098	81 620
2	65	266	318	17	153	68 098	81 621
3	108	532	636	18	101	68 099	81 621
4	29	1064	1272	19	177	136 198	163 242
5	27	1064	1273	20	173	136 199	163 242
6	56	1064	1274	21	161	136 200	163 242
7	83	1064	1275	22	125	136 201	163 242
8	87	2128	2550	23	17	136 202	163 242
9	51	4256	5100	24	22	136 202	163 243
10	66	4256	5101	25	39	136 202	163 244
11	60	8512	10 202	26	61	136 202	163 245
12	20	17 024	20 404	27	141	272 404	326 490
13	68	17 024	20 405	28	65	272 405	326 490
14	149	34 048	40 810				

由表 4.6 可以看出 $x_{28}=x_2$，且 $\gcd(89,b_{28}-b_2)=1$，则根据

$$x(b_{28}-b_2)=a_2-a_{28}\pmod{89}$$

来得到 x。令：

$$r=(b_{28}-b_2)(\bmod 89)=76$$

则：

$$r^{-1}=76^{-1}(\bmod 89)=41$$
$$x=41*(a_i-a_j)(\bmod 89)$$
$$=41*(266-272\,405)(\bmod 89)$$
$$=53$$

所以 $x=\log_3 75=53$。

以上对攻击方法针对任意的离散对数问题。对于椭圆曲线离散对数问题，有时由于参数选取不当，形成特定的安全性较弱的特殊椭圆曲线，此时可能存在亚指数时间复杂度甚至多项式复杂度的求解算法。1991 年由 Menezes、Okamoto 和 Vanstone 提出的 MOV 攻击方法及 1998 年 Smart、Semaev、Satoh 和 Araki 提出的 SSAS 算法就是这种情形下的典型攻击方法。感兴趣的读者可以查阅相关文献。

随着 ECC 的不断完善，ECC 逐渐发展成当今密码体制中最受青睐的公钥密码体制之一。1998 年 ECC 被确定为 ISO/IEC 数字签名标准 ISO14888-3；1999 年椭圆曲线数字签名算法 ECDSA 被 ANSI 机构接纳为数字签名标准 ANSI X9.62，同时椭圆曲线 DH 体制版本 ECDH 被确定为 ANSI X9.63；2000 年 ECC 被确定为 IEEE 标准的 IEEE 1363—2000。

4.7　习　题

1. 找出乘群 $(Z_{13}^*, *)$ 中所有的生成元。
2. 找出加群 $(Z_{12}, +)$ 中所有的生成元。
3. 试求群 $(Z_{13}^*, *)$ 中元素 3 和 7 的阶。
4. 试求群 $(Z_{12}, +)$ 中元素 3 和 7 的阶。
5. 试求群 $(Z_{12}, +)$ 中所有元素的阶。
6. 已知下列 4 个置换

$$\pi_1=\begin{bmatrix}1&2&3&4\\1&2&3&4\end{bmatrix},\quad \pi_2=\begin{bmatrix}1&2&3&4\\1&2&4&3\end{bmatrix}$$
$$\pi_3=\begin{bmatrix}1&2&3&4\\2&1&3&4\end{bmatrix},\quad \pi_4=\begin{bmatrix}1&2&3&4\\2&1&4&3\end{bmatrix}$$

组成的群 G，试写出 G 的乘法表，并且求出 G 的单位元及 π_1^{-1}、π_2^{-1}、π_3^{-1}、π_4^{-1} 和 G 的所有子群。

7. 设 a 和 b 是一个群 G 的两个元且 $ab=ba$，又设 a 的阶 $|a|=m$，b 的阶 $|b|=n$，并且 $(m,n)=1$，证明：ab 的阶 $|ab|=mn$。

8. 有限域上的椭圆曲线 E：$y^2=x^3+10x+22(\bmod 31)$：

(1) 求椭圆曲线上所有的点；

(2) 求点 $(2,9)$ 的阶；

(3) 求 $9(7,1)$；

(4) 若 $k(9,2)=(2,9)$，求 k。

9. 使用 ElGamal 公钥密码算法对信息 $m=2005$ 进行加密，ElGamal 公钥体制的参数是 $p=2999$，$\alpha=11$，$a=168$，$k=296$。

10. 使用 ElGamal 公钥密码算法对信息 $c=(831,1275)$进行解密，ElGamal 公钥体制的参数是 $p=3001$，$\alpha=19$，$a=158$，$k=2005$。

11. 根据群的定义，证明有限域上的椭圆曲线上的点和运算“+”构成 Abel 群。

12. 求椭圆曲线 E：$y^2=x^3+10x+22(\text{mod } 31)$上的点。

13. 在椭圆曲线 E：$y^2=x^3+10x+22(\text{mod } 31)$上，已知 P、Q、R 3 点：

$$P=(2,9),\quad Q=(6,9),\quad R=(4,8)$$

分别求 $P+Q$、$Q+R$。

14. 在椭圆曲线 E：$y^2=x^3+10x+22(\text{mod } 31)$上，已知 P、Q、R 3 点：

$$P=(20,10),\quad Q=(18,19),\quad R=(21,10)$$

试验证：

$$(P+Q)+R=P+(Q+R)$$

15. 椭圆曲线 E：$y^2=x^3+10x+22(\text{mod } 31)$，取点 P：$P=(9,2)$，分别求 $2P$、$5P$、$17P$、$23P$ 和 $34P$。

16. 椭圆曲线 E：$y^2=x^3+10x+22(\text{mod } 31)$，找出所有生成元。

17. 椭圆曲线 E：$y^2=x^3+10x+22\ (\text{mod } 31)$，分别求点 $P=(20,10)$、$Q=(18,19)$、$R=(21,10)$的阶。

18. 利用椭圆曲线 E：$y^2=x^3+10x+22(\text{mod } 31)$，设计一种 MV 椭圆曲线公钥密码系统。

19. 使用椭圆曲线公钥密码算法对信息 $m=(22,26)$进行加密，椭圆曲线公钥体制的参数是 E：$y^2=x^3+10x+22(\text{mod } 31)$，$\alpha=(7,1)$，$\beta=(2,9)$，$a=19$，$t=5$。

20. 使用椭圆曲线公钥密码算法对信息 $m=((29,26),2,30)$进行解密，椭圆曲线公钥体制的参数是 E：$y^2=x^3+10x+22(\text{mod } 31)$，$\alpha=(7,1)$，$\beta=(2,9)$，$a=19$，$t=5$。

第5章　其他公钥密码系统

除了基于大数因式分解难题或离散对数问题的公钥密码系统之外，公钥密码系统大家族中还包含许多其他的公钥密码系统。这些其他的公钥密码系统各具特色，基于各种类型的 NP 问题。

5.1　背包公钥系统

背包公钥密码系统建立在子集和问题的基础之上，以下称背包问题。

背包问题的一般描述如下。

假设有 N 件物品和一个背包，不妨设其中第 i 件物品的重量是 c_i，价值是 w_i，W 是背包能够承受的最大重量。如果要从 N 件物品中取几件物品，装进背包，问题是将哪几件物品装入背包可使这些物品的重量总和不超过背包容量 V，且总价值最大。

原则上只要搜索集合 $C=\{c_1,c_2,\cdots,c_N\}$ 的所有子集，并检验子集中元素之和是否不超过 V，把所有满足条件的子集找出来，然后从中挑出总价值最大的一个子集，问题就能解出。

显然要搜索的子集个数为：

$$C_N^0 + C_N^1 + \cdots + C_N^N = 2^N$$

现有的结论是：背包问题是 NP 完全问题。

在理论上，很多整数规划问题的解决都能够借助背包问题得以解决；在实际应用中，如资源分配、投资决策、货物运输等应用都可用背包问题来建模。

1978 年，Merkle 和 Hellman 率先提出利用背包问题建立公钥密码体制，称为 MH 背包公钥密码系统(Merkle-Hellman Knapsack System)。

不过，MH 背包公钥密码系统中的背包问题是另一类背包问题，与背包问题的一般描述略有不同。

另一种背包问题可描述如下：

假设有 N 件物品和一个背包，不妨设其中第 i 件物品的重量是 c_i，问题是如何从 N 件物品中取几件物品使得这几件物品的总重量等于 V。

由于要搜索的子集个数依然是 2^N，所以此背包问题也是 NP 完全问题。此背包问题又称子集和问题。

为了构造公钥体制，下面通过引入一个超递增性序列，形成一类特殊的子集和问题。

超递增性序列的子集和问题：已知集合 $I=(s_1,\cdots,s_n,z)$，其中 s_1、$\cdots$、s_n 是满足超递增性的序列，即：

$$s_j > \sum_{i=1}^{j-1} s_i \quad 对于\ 2 \leqslant j \leqslant n$$

z 为目标和，问是否存在 $x=(x_1,\cdots,x_n)$(x_i 的取值只能是 0 或 1)使得：

$$\sum_{i=1}^{n} x_i s_i = z$$

简单地说，所谓超递增性是指每一个靠后的数都比前面所有数的和还大。例如，发行的人民币的序列是(单位元)0.01、0.02、0.05、0.1、0.2、0.5、1、2、5、10、20、50、100，不难验证：

$$0.01+0.02<0.05$$
$$0.01+0.02+0.05<0.1$$
$$0.01+0.02+0.05+0.1<0.2$$
$$\vdots$$

说明人民币序列满足超递增性。

虽然背包问题是 NP 完全问题，但是超递增性序列的子集和问题的求解却相对容易。

此类背包问题的典型求解算法是“贪心”算法(Greedy Algorithm)。

首先选最大的数 s_n 放入背包，若能放入，令相应的 x_n 为 1；否则，令相应的 x_n 为 0。再从 s 中减去 $x_n s_n$ 后求 x_{n-1}，以此类推，直到求出 x_1 为止。当然，也可能无解。

具体算法描述如下：

```
r = z
for k = n to 1 do
      if r >= s(k) then
              r = r - s(k)
              x(k) = 1
        else
              x(k) = 0
if ∑(k=1..n) x(k)s(k) = z then
        x = (x(1),x(2), … ,x(n))就是背包问题的解
else
        该背包问题无解
```

此类子集和问题能够求解，恰恰是利用了超递增性序列的特性。

例 5.1 求解背包问题：(2,5,9,21,45,103,215,450,946 | 1643)。

通过背包问题的贪心算法来求解：

$r(0)=1643$

第 1 步： $1643>846$ $x(8)=1$ $r(1)=1643-946=697$

第 2 步： $697>450$ $x(7)=1$ $r(2)=247$

第 3 步： $450>215$ $x(6)=1$ $r(3)=32$

第 4 步： $32<103$ $x(5)=0$ $r(4)=32$

第 5 步： $32<45$ $x(4)=0$ $r(5)=32$

第 6 步： $32>21$ $x(3)=1$ $r(6)=11$

第 7 步： $11>9$ $x(2)=1$ $r(7)=2$

第 8 步： $2<5$ $x(1)=0$ $r(8)=2$

第 9 步： $2=2$ $x(1)=1$ $r(9)=0$

于是最终解为：

$$x=(1,0,1,1,0,0,1,1,1)$$

背包公钥密码系统：设 $s=(s_1,\cdots,s_n)$是满足 $s_j>\sum_{i=1}^{j-1}s_i$ 的超递增序列，取满足 $p>\sum_{i=1}^{n}s_i$ 的素数 p，和满足 $1\leqslant a\leqslant p-1$ 的 a，计算：

$$t=(t_i,\cdots,t_n),\quad t_i=as_i \bmod p$$

明文空间 $P=\{0,1\}^n$，密文空间 $C=\{0,\cdots,n(p-1)\}$

密钥 $K=\{(s,p,a,t)\}$，其中，t 为公钥，s、p、a 是私钥。

加密算法：

$$e_K(x_i,\cdots,x_n)=\sum_{i=1}^{n}x_it_i$$

解密算法：

$$z=a^{-1}y \bmod p$$

背包问题$(s_1,\cdots,s_n|z)$的解$(x_i,\cdots,x_n)$就是解密结果。

例 5.2　在背包公钥密码系统中，设 $s=(2,5,9,21,45,103,215,450,946)$，$p=2003$，$a=1289$，根据 s、p 和 a 可计算出 t：

$$t=(575,436,1586,1030,1921,569,721,1183,1570)$$

t 是公钥，用 t 对消息 $x=(1,0,1,1,0,0,1,1,1)$进行加密：

$$a=575+1586+1030+721+1183+1570=6665$$

解密时，先求 z：

$$z=1289^{-1}6665 \bmod 2003=1643$$

得到背包问题$(2,5,9,21,45,103,215,450,946\mid 1643)$，根据前面的贪心算法，可解出 x：

$$x=(1,0,1,1,0,0,1,1,1)$$

5.2　McEliece 公钥密码系统

1978 年，Berlekamp、McEliece 和 VanTilborg 证明了一般线性纠错码的译码问题是 NP 完全问题。紧接着，McEliece 利用这一 NP 完全问题，首次构造出了一类基于一般线性纠错码的公钥密码体制，称为 McEliece 公钥体制。

介绍 McEliece 公钥体制前，有必要先了解一些关于编码的基础知识。

在数字信号传输过程中，由于噪声的存在及通信信道特性不理想，都可使信号波形失真，其结果是在接收端接收到错误的编码。为了尽可能提高数字通信的可靠性，引入差错控制编码，又称纠错编码。

纠错编码分以下两步实施：

(1) 编码，即在要传送的数字信息序列中按一定规则混入一些冗余码元(校验位)。编码过程也可以理解成一个变换。

通常以(n,l)组码表示由 l 位 0、1 字符串转换成 n 位 0、1 字符串的编码过程：

将信息

$$m=m_1m_2\cdots m_l\quad m_i\in\{0,1\}\quad i=0,1,\cdots,l$$

变换为码文

$$E(m) = w = w_1 w_2 \cdots w_n \quad w_i \in \{0,1\} \quad i = 0,1,\cdots,n$$

(2) 译码，即在接收端鉴别传输过程是否发生错误或纠正错误，恢复原始信息序列。

例 5.3　奇偶校验码可表示成$(n+1,n)$检错码的形式，即在 n 位信息后面附加一位奇偶校验位：

$$E(m) = m = m_1 m_2 \cdots m_n m_{n+1}$$

如果传输过程中出现了一个错位，即将某一位的 0 错为 1 或将 1 错为 0 则可立即被发现。当然如果 n 位中出现两个错位，反而发现不了。奇偶校验码并不能判定哪位出了差错，因而只能是检错码。

与奇偶校验码不同，重复码可用来纠错。

采用重复 3 次的方式进行编码，即$(3n,n)$码：

$$E(m) = m_1 m_2 \cdots m_n \, m_1 m_2 \cdots m_n \, m_1 m_2 \cdots m_n$$

即对每组位的码连续传输了 3 遍。如果某一位出现不同，而另外两组是一样的便以出现两次一样的为准。当然，同时在同一位上出现两次或三次错的可能性依然存在，但出现概率就少多了。

设一次传输出错概率为 $P_e=0.001$，则$(3n,n)$码传输正确的概率为：

$$(1-0.001)^3 + 3 * 0.001 * (1-0.001)^2 = 0.997\,003 + 0.002\,994 = 0.999\,997$$

所以传输正确的概率从单次传输的 0.999 提高到 0.999 997，付出的代价是码文比明文扩大了 3 倍。

纠错编码有多种分类：按功能分，可分为检错码和纠错码；按校验码元与信息码元之间是否存在线性关系，可分为线性码与非线性码；按信息码元与校验码元之间的约束关系不同，可分为分组码与非分组码（卷积码）；按码元的取值方法，可分为二进制码与多进制码。

定义(**Hamming 重量**)：已知 $m=m_1 m_2 \cdots m_n, m_i \in \{0,1\}, i=1,\cdots,n$，定义

$$w(m) = \sum_{i=1}^{n} m_i$$

为 m 的 Hamming 重量或权，也就是 m 的 n 位中 1 的数量。

定义(**Hamming 距离**)：已知：

$$a = (a_1, a_2, \cdots, a_{n-1}) \quad a_i \in \{0,1\}, i = 1,2,\cdots,n$$

$$b = (b_1, b_2, \cdots, b_{n-1}) \quad b_i \in \{0,1\}, i = 1,2,\cdots,n$$

定义 $d(a,b)$表示 a 和 b 各对应码元之间不相同的个数，即：

$$d(a,b) = \sum_{i=1}^{n} | a_i - b_i | = \sum_{i=1}^{n} (a_i \oplus b_i)$$

关于 Hamming 距离有以下的结论，此处不加证明。

若 a、b、c 都是长度为 n 的 0、1 字符串：

$$a = (a_1, a_2, \cdots, a_n) \quad a_i \in \{0,1\}, i = 1,\cdots,n$$

$$b = (b_1, b_2, \cdots, b_n) \quad b_i \in \{0,1\}, i = 1,\cdots,n$$

$$c = (c_1, c_2, \cdots, c_n) \quad c_i \in \{0,1\}, i = 1,\cdots,n$$

则：

$$d(a,b) = d(b,a)$$
$$d(a,c) \leqslant d(a,b) + d(b,c)$$

定理 5.1　一组码可以检出 k 个错误的充要条件是码间最短距离至少为 $k+1$。

证明：设 $w=w_1w_2\cdots w_n$ 是码字，w 传输中有误差 e，收到的是 $r=w+e$，e 能被检出的充要条件是 $w+e$ 不是码字，因此能检出 k 个错误的充要条件是码间最短距离至少为 $k+1$。

定理 5.2　如果两个码字组之间的最短距离为 $2k+1$ 则可以纠正不超过 k 的误码。

证明：设码字 a 传输过程中发生错误得到的是 r，且 $d(a,r)\leqslant k$ 则不存在别的码字与 r 的距离不超过 k。否则设该码字为 b，满足 $d(b,r)\leqslant k$，于是有：

$$d(a,b) \leqslant d(a,r) + d(b,r) \leqslant k + k$$

这与假设产生矛盾，命题得证。

以上两个定理给出了检错码和纠错码的构造要求。

例 5.4　已知码字：

10010　01001　10101　01110

两个码字之间进行模 2 加运算，可得表 5.1 模 2 加运算表。

表 5.1　模 2 加运算表

	10010	01001	10101	01110
10010	—	11011	00111	11100
01001	11011	—	11100	00111
10101	00111	11100	—	11011
01110	11100	00111	11011	—

由表 5.1 可见，任意两个码字的最小距离是 3，因此能纠正 1 个错误。

具体纠错的译码表，如表 5.2 所示。

表 5.2　纠错译码表

w	10010	01001	10101	01110
r	00010	11001	00101	11110
	11010	00001	11101	00110
	10110	01101	10001	01010
	10000	01011	10111	01100
	10011	01000	10100	01111

表 5.2 中给出的是距离为 1 的译码表，如 $r=11001$，由于它与 01001 的距离为 1，因此译为 01001，并且能够发现错误发生在第 1 位上。

可以引入生成矩阵来规范编码过程。

$$\text{令 } G = (g_{ij})_{l\times n} = \begin{bmatrix} G_1 \\ G_2 \\ \vdots \\ G_{l-1,1} \\ G_l \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1,n-1} & g_{1,n} \\ g_{21} & g_{22} & \cdots & g_{2,n-1} & g_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ g_{l-1,1} & g_{l-1,2} & \cdots & g_{l-1,n-1} & g_{l-1,n} \\ g_{l1} & g_{l2} & \cdots & g_{l,n-1} & g_{1n} \end{bmatrix}$$

其中，$g_{ij}\in\{0,1\}$，$i=1,2,\cdots,l$；$j=1,2,\cdots,n$。

有了生成矩阵，编码过程可通过矩阵运算实现，即：

$$W = E(m) = (m_1 \quad m_2 \quad \cdots \quad m_{l-1} \quad m_l)\begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1,n-1} & g_{1,n} \\ g_{21} & g_{22} & \cdots & g_{2,n-11} & g_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ g_{l-1,1} & g_{l-1,2} & \cdots & g_{l-1,n-1} & g_{l-1,n} \\ g_{l1} & g_{l2} & \cdots & g_{l,n-1} & g_{1n} \end{bmatrix}$$

$$= \sum_{i=1}^{l} m_i G_i$$

例 5.5　取：

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

则信息 110 对应的码字是：

$$W = (1 \quad 1 \quad 0)\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

$$= (1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1)$$

现将长度为 3 的信息和对应的码字列表，如表 5.3 所示。

表 5.3　信息和对应的码字表

信息			码字					
0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	0	1
0	1	0	0	1	0	0	1	1
0	1	1	0	1	1	1	1	0
1	0	0	1	0	0	1	1	0
1	0	1	1	0	1	0	1	1
1	1	0	1	1	0	1	0	1
1	1	1	1	1	1	0	0	0

表 5.3 中码字（共 6 位）的前 3 位为信息位，余下的 3 位是校验位。

同理，如果是 (n,l) 码，可将 G 写成以下形式：

$$G = \left[\begin{array}{ccccc|ccccc} 1 & 0 & \cdots & 0 & 0 & g_{11} & g_{12} & \cdots & g_{1,n-l-1} & g_{1,n-l} \\ 0 & 1 & \cdots & 0 & 0 & g_{21} & g_{22} & \cdots & g_{2,n-l-1} & g_{2,n-l} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & g_{l-1,1} & g_{l-1,2} & \cdots & g_{l-1,n-l-1} & g_{l-1,n-l} \\ 0 & 0 & \cdots & 0 & 1 & g_{l1} & g_{l2} & \cdots & g_{l,n-l-1} & g_{1,n-l} \end{array}\right]$$

于是，m 的码字是：

$$W = (m_1 m_2 \cdots m_l) \quad G = (m_1 m_2 \cdots m_l m_{l+1} \cdots m_n)$$

其中：

$$m_{l+j} = \sum_{i=1}^{l} g_{ij} m_i, \quad j = 1,2,\cdots,n-l$$

可将以上等式换成另一种形式：

$$\begin{bmatrix} g_{11} & g_{21} & \cdots & g_{l-1,1} & g_{l1} & 1 & 0 & \cdots & 0 & 0 \\ g_{12} & g_{22} & \cdots & g_{l-1,2} & g_{l2} & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ g_{1,n-l-1} & g_{2,n-l-1} & \cdots & g_{l-1,n-l-1} & g_{l,n-l-1} & 0 & 0 & \cdots & 1 & 0 \\ g_{1,n-l} & g_{2,n-l} & \cdots & g_{l-1,n-l} & g_{1,n-l} & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

即：

$$Hr^T = O$$

此方程又称**校验方程**，H 称为**校验矩阵**。

校验矩阵和生成矩阵的关系是：

$$G = (I_l \mid A)_{l\times n} \quad A = (a_{ij})_{l\times(n-l)}, I_l \text{ 为 } l \text{ 阶方阵}$$

$$H = (A^T \mid I_{n-l})_{(n-l)\times n}$$

校验矩阵的主要用途是纠错。

设 $r=W+e$，e 表示差错：

$$Hr^T = H(W+e)^T = HW^T + He^T = He^T$$

如果 $He^T \neq 0$，可由 He^T 看出错误出现在第几位上，予以纠正。

例 5.6　已知 $H=\begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$，$r=(1\quad 0\quad 0\quad 1\quad 0\quad 1)$

由于：

$$Hr^T = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

$(0\quad 1\quad 1)^T$ 正好是 H 矩阵中的第 2 列，所以可判定第 2 位出错，于是将第 2 位由 0 改为 1，得到：

$$W = 110101$$

信息位是 110，110 就是正确的译码。

由以上的译码过程还可以得到定理 5.3。

定理 5.3　$(n-l)\times n$ 的校验矩阵能正确纠正 1 个错误的充要条件是 H 的各列为不相同的非零列向量。

依据此定理，当 k 确定以后 n 应取 2^k-1，因为 2^k-1 个列向量覆盖所有的非零状态。

例 5.7　当 $k=3$ 时，$n=2^3-1=7$，$1=7-3=4$，此时：

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

H 包含除$(0\quad 0\quad 0)^T$ 以外的所有状态，能够纠正 1 个错误。对应的生成矩阵为：

$$G=\begin{bmatrix}1&0&0&0&0&1&1\\0&1&0&0&1&0&1\\0&0&1&0&1&1&0\\0&0&0&1&1&1&1\end{bmatrix}$$

用这种方法构成的可纠一个错的码称为(7,4)汉明码(Hamming Code)。

汉明码是纠一位差错的纠错码,它的每一位差错都对应于 H 中的某一列。如果出现两位差错,其差错对应于相应的两列之和,这两列之和又等于另外的某一列,故无法实现纠错。

20 世纪 70 年代初,俄国学者 Goppa 系统地构造出了一类有理分式码:Goppa 码。Goppa 码的主要优点是它的某些子类能逼近香农编码理论给出的最佳性能。

设 $0<n\leqslant q^m$,$L=\{a_0,a_1,\cdots,a_{n-1}\}$是一个有序集合,称为位置集,$a_i\in GF(q^m)$且对任何 $i\neq j$ 有 $a_i\neq a_j$,i、$j\leqslant n$,又设 $GF(q)$上的 n 维线性空间 $GF^n(q)$,码字:

$$C=(c_{n-1},c_{n-2},\cdots,c_0)\in GF^n(q)$$

则与 C 对应的 $GF(q^m)$上的 z 有理分式:

$$R_C(z)=\sum_{i=0}^{n-1}\frac{c_i}{z-a_i}$$

又设 $g(z)$是系数在 $GF(q^m)$上的 r 次 z 多项式,它的根不在位置集 L 中,则以下 n 重集合:

$$\{C:R_C(z)=0 \bmod g(z),\ C\in GF^n(q)\}$$

称为由 $g(z)$生成的 **Goppa 码**,用 $\Gamma(g(z),L)$表示;称 $g(z)$是 Goppa 码的生成多项式或 Goppa 多项式。若 $g(z)$在 $GF(q^m)$上既约,则 $\Gamma(g(z),L)$称为**既约 Goppa 码**。

由定义看到,Goppa 码 $\Gamma(g(z),L)$是 n 维线性空间中,满足上式 n 重集合,显然它是一个线性子空间,因而 Goppa 码是一个线性码。

位置集和既约多项式的选取可以参考以下的例子。

例 5.8 设 α 满足 $GF(2^4)$上生成多项式 $x^4+x+1=0$,可得 α 的模指数表,如表 5.4 所示。

表 5.4 α 的模指数运算表

1	0	0	0	1
α	0	0	1	0
α^2	0	1	0	0
α^3	1	0	0	0
α^4	0	0	1	1
α^5	0	1	1	0
α^6	1	1	0	0
α^7	1	0	1	1
α^8	0	1	0	1
α^9	1	0	1	0
α^{10}	0	1	1	1
α^{11}	1	1	1	0
α^{12}	1	1	1	1
α^{13}	1	1	0	1
α^{14}	1	0	0	1
0	0	0	0	0

取 $L=\{0,1,\alpha,\alpha^2,\alpha^3,\alpha^4,\alpha^5,\alpha^6,\alpha^7,\alpha^8,\alpha^9,\alpha^{10},\alpha^{11},\alpha^{12},\alpha^{13},\alpha^{14}\}$,并取:

$$g(z)=z^2+z+\alpha^3$$

显然在 $GF(2^4)$中 $g(z)$是既约的，依次验证可知 L 中不存在 $g(z)$的根。于是，可由 L 和 $g(z)$生成 Goppa 码 $\Gamma(g(z),L)$。

由 Goppa 码的定义可知：

$$R_C(z)=\sum_{i=0}^{n-1}\frac{c_i}{z-a_i}=0 \bmod g(z)$$

该式还可以写成：

$$\left(\frac{1}{z-a_0}\quad \frac{1}{z-a_1}\quad \cdots \quad \frac{1}{z-a_{n-1}}\right)\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix}=H_1C^T=0 \bmod g(z)$$

根据前面校验矩阵的定义，H_1就是 Goppa 码的校验矩阵。

例 5.9　已知 α、L、$g(z)$的参数与前面的例子相同，求 Goppa 码 $\Gamma(g(z),L)$的校验矩阵。

解：将 L 和 $g(z)$带入前面的 Goppa 码 $\Gamma(g(z),L)$的校验矩阵中，有：

$$H_1=\left(\frac{1}{z-0}\quad \frac{1}{z-1}\quad \cdots \quad \frac{1}{z-\alpha^{14}}\right) \bmod (z^2+z+\alpha^3)$$

可用欧几里德扩展算法求解$\frac{1}{z-\omega} \bmod (z^2+z+\alpha^3)$，其结果如表 5.5 所示。

表 5.5　$z-\omega$ 的模逆运算结果表

ω	0	1	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
$g_0(\omega)$	α^{14}	0	α^{10}	α^3	α^{10}	α^9	α^{13}	1	α^9	α^{13}	α^{11}	α^8	α^{11}	α^{14}	α^3	α^8
$g_1(\omega)$	α^{14}	α^{14}	α^{13}	α^9	α^6	α^6	α^3	α^7	α^{11}	α^7	α^9	α^3	α^{12}	α^{13}	α^{11}	α^{12}

在表 5.5 中，$\frac{1}{z-\omega} \bmod (z^2+z+\alpha^3)=g_0(\omega)+g_1(\omega)z$，如：

$$\frac{1}{z-\alpha^3} \bmod (z^2+z+\alpha^3)=\alpha^{10}+\alpha^6 z$$

由 $H_1C^T=0 \bmod g(z)$可得：

$$(1\quad z)\begin{pmatrix} g_0(\omega) \\ g_1(\omega) \end{pmatrix}C^T=0 \bmod g(z)$$

于是 Goppa 码 $\Gamma(g(z),L)$的校验矩阵可写成以下形式：

$$\begin{pmatrix} g_0(\omega) \\ g_1(\omega) \end{pmatrix}$$

将 $g_0(\omega)$和 $g_1(\omega)$的具体编码带入可得：

$$H=\begin{bmatrix}
0&0&0&0&0&1&0&1&1&0&1&0&1&0&0&0\\
0&0&1&0&1&0&1&0&0&1&0&1&0&0&0&1\\
1&0&0&0&0&1&1&0&1&1&1&1&1&1&0&1\\
1&0&1&1&1&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&1&1&1&0&1&1&1&1&0&1&0&1&1\\
0&0&1&0&1&1&0&1&0&1&0&0&1&1&0&1\\
1&1&1&1&1&1&0&1&1&1&1&0&0&1&1&0\\
1&1&0&0&1&1&1&0&1&0&0&1&0&0&1&0
\end{bmatrix}$$

以上就是 Goppa 码 $\Gamma(g(z),L)$ 的校验矩阵。

Goppa 码的校验矩阵中任取 r 列构成的矩阵的秩为 r 且该矩阵中各行线性无关，任意 r 列也线性无关。由线性分组码理论可知，由 H_1 或 H 矩阵确定的 Goppa 码的最小距离：

$$d_{\min} \geqslant r+1$$

所以，Goppa 码的纠错能力是 $[r/2]$（不超过 $r/2$ 的正整数）。

1978 年 McEliece 用 Goppa 码构造了一类公钥密码体制，开辟了用纠错码构造公钥密码体制的先河。

McEliece 公钥密码体制： 在 $GF(2^l)$ 上随机选取一个 t 次既约多项式 $g(x)$，由 $g(x)$ 得到一个码长 $n=2^l$，维数 $k \geqslant n-tl$ 的既约 Goppa 码。

设 G 是二元 Goppa 码的生成矩阵（$k\times n$ 阶），校验矩阵是（$n-k$）$\times n$ 阶矩阵 H。

随机选取一个 $GF(2^l)$ 上的 $k\times k$ 阶非奇异矩阵 S 及 $n\times n$ 阶置换矩阵 P，计算 $G'=SGP$。

将 G' 公开作为公钥，S、G 和 P 是私钥。

由 G' 代表的码字与由 G 代表的码字组合等价，G' 代表的是一般的线性分组码。

加密算法：

将明文消息 m 分为长是 k 的位串，用公钥 G' 计算密文向量：

$$c = mG' + e$$

这里，e 是 n 比特的二元随机误差向量。

解密算法：

(1) 计算 $y_1=cP^{-1}$；

(2) 由 $y_1=m_1+e_1$，$m_0G=m_1$，译码求得 m_0；

(3) 计算 $m=m_0S^{-1}$。

下面以一个参数较小的例子加以说明。

例 5.10 已知 α、L、$g(z)$ 的参数与前面的例子相同，H 是 Goppa 码 $\Gamma(g(z),L)$ 的校验矩阵。

生成矩阵可通过前面例子中的 H 求出：

$$G = \begin{bmatrix}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&1\\
1&0&1&0&1&1&0&1&0&0&0&0&0&0&1&0\\
1&0&0&1&1&1&1&1&0&0&0&0&0&1&0&0\\
0&0&1&1&1&1&0&0&0&0&0&0&1&0&0&0\\
0&1&1&1&1&0&1&0&0&0&0&1&0&0&0&0\\
0&1&1&0&0&0&1&1&0&0&1&0&0&0&0&0\\
0&1&1&1&0&1&0&1&0&1&0&0&0&0&0&0\\
1&0&1&0&0&1&1&0&1&0&0&0&0&0&0&0
\end{bmatrix}$$

随机选取：

$$S=\begin{bmatrix}1&1&1&1&1&1&1&1\\0&1&1&1&1&1&1&1\\0&0&1&1&1&1&1&1\\0&0&0&1&1&1&1&1\\0&0&0&0&1&1&1&1\\0&0&0&0&0&1&1&1\\0&0&0&0&0&0&1&1\\0&0&0&0&0&0&0&1\end{bmatrix}$$

选取置换矩阵：

$$P=\begin{bmatrix}1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\end{bmatrix}$$

于是可得：

$$G'=SGP=\begin{bmatrix}0&1&0&1&1&0&1&0&0&1&1&1&1&1&1&1\\1&0&1&1&0&0&1&0&0&1&0&1&1&1&1&1\\0&1&1&0&0&1&1&0&1&1&0&1&1&1&1&0\\1&1&1&1&1&0&1&1&0&1&0&1&1&1&0&0\\1&0&1&0&0&1&1&1&0&1&0&1&1&0&0&0\\1&1&0&0&1&0&1&0&0&1&0&1&0&0&0&0\\1&0&1&0&1&0&1&1&1&1&0&0&0&0&0&0\\1&1&0&1&0&0&0&1&0&1&0&0&0&0&0&0\end{bmatrix}$$

将 G' 公开作为公钥，S、G 和 P 是私钥。

已知明文 $m=(1\quad 1\quad 0\quad 0\quad 1\quad 0\quad 0\quad 1)$，并假设：

$$e=(0\quad 0\quad 0\quad 0\quad 1\quad 0\quad 0\quad 0\quad 1\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0)$$

加密得到密文：

$$\begin{aligned}c&=mG'+e\\&=(1\quad 0\quad 0\quad 1\quad 0\quad 1\quad 1\quad 0\quad 1\quad 0\quad 1\quad 1\quad 1\quad 0\quad 0\quad 0)\end{aligned}$$

解密时先计算：

$$cP^{-1} = (1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1)$$

对 cP^{-1} 进行快速译码得：

$$m_0 = (1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0)$$

最后解出明文：

$$m = m_0 S^{-1} = (1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1)$$

利用 Goppa 码构造公钥算法的方法可以推广到其他的一般线性码，如汉明码和 BCH 码等。

下面以(7,4)汉明码为例，进一步说明 McEliece 公钥算法的思路。

例 5.11 已知

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

是(7,4)汉明码的生成矩阵，首先选取矩阵：

$$S = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

并选取置换矩阵：

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

然后，可得公钥矩阵：

$$G' = SGP = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

接下来对明文 $m=(1 \quad 1 \quad 0 \quad 1)$加密，选取误差向量 $e=(0,0,0,0,1,0,0)$，可得：

$$\begin{aligned} c &= mG' + e \\ &= (1 \quad 1 \quad 0 \quad 1)\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} + (0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0) \\ &= (0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0) \end{aligned}$$

解密时，现通过 c 计算：

$$y_1 = cP^{-1} = (0\quad 1\quad 1\quad 0\quad 1\quad 1\quad 0)\begin{bmatrix} 0&0&0&1&0&0&0\\ 1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0\\ 0&1&0&0&0&0&0\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&1&0\\ 0&0&1&0&0&0&0 \end{bmatrix}$$

$$= (1\quad 0\quad 0\quad 0\quad 1\quad 1\quad 1)$$

于是，根据 e 和 P^{-1} 可得到：

$$m_1 = y_1 + eP^{-1} = (1\quad 0\quad 0\quad 0\quad 1\quad 1\quad 0)$$

接下来取前面 4 个分量：

$$m_0 = (1\quad 0\quad 0\quad 0)$$

最后还原出明文：

$$m = S^{-1}m_0 = \begin{bmatrix} 1&1&0&1\\ 1&0&0&1\\ 0&1&1&1\\ 1&1&0&0 \end{bmatrix}(1\quad 0\quad 0\quad 0)$$

$$= (1\quad 1\quad 0\quad 1)$$

以上两个例子足以说明，McEliece 公钥密码体制并非只能使用 Goppa 码，它代表的是一类基于线性纠错码的公钥密码体制。

5.3　NTRU 公钥密码系统

NTRU(Number Theory Research Unit)公钥密码体制是在 20 世纪 90 年代中期，由美国数学家 Hoffstein 博士首先在 CRTPTO'96 会议上提出并命名的。很快 NTRU 被接受为 IEEE P1363 标准，并被标准化在文档 Working Group for Standards In Public-KeyCryptography 中。

NTRU 是一种在截尾多项式环上构造的密码体制，其安全性依赖于格中的最短向量问题(Shortest Vector Problem，SVP)。

与 RSA 算法相比，NTRU 算法具有以下优势：

(1) 速度快。NTRU 加密、解密一个长度 N 的信息分组需要 $O(N^2)$次操作，而 RSA 需要 $O(N^3)$次操作。而且 RSA 的基本运算往往是上千位的求模指数，而 NTRU 一般仅是小于 255 位的运算；所以 NTRU 的运算速度相对较快。

(2) 系统需求低。NTRU 比 RSA 在处理器能力和硬件资源上的需求要小，并且更易于用硬件实现。

(3) 密钥生成快。NTRU 中随机选择多项式，并支持将长密钥拆成小块，提高了密钥的生成速度。

NTRU 算法的这些特点使其在嵌入式系统中应用广泛。

下面先介绍 NTRU 公钥密码体制中用到的一些预备知识。

定义 5.1(环) 有两个二元运算(分别称为加法,乘法)的代数系(A,+,×)称为一个环,如果:

(1) $(A,+)$是一个加法群;

(2) $(A,\times)$是一个乘法半群;

(3) 乘法对于加法的左、右分配率都成立,即:

$$a(b+c)=ab+ac$$
$$(b+c)a=ba+ca$$

对于 A 中,任意 a、b、c 均成立。

定义 5.2(截尾多项式环) 考虑整系数 $N-1$ 次多项式:

$$a=a_0+a_1x+\cdots+a_{N-1}x^{N-1}$$

其系数$(a_0,a_1,\cdots,a_{N-1})$为整数,所有这样的多项式构成集合 R,令:

$$b=b_0+b_1x+\cdots+b_{N-1}x^{N-1}$$

在 R 中定义两种运算:

(1) 加法运算:

$$a+b=(a_0+b_0)+(a_1+b_1)x+\cdots+(a_{N-1}+b_{N-1})x^{N-1}$$

(2) 乘法运算(卷积,Cyclic Convolution Product):

$$a\otimes b=c_0+c_1x+\cdots+c_{N-1}x^{N-1}$$
$$c_k=\sum_{i+j=k \bmod N} a_ib_j,\quad k=0,1,\cdots,N-1$$

根据环的定义,$(R,+,\otimes)$构成环,称为截尾多项式环(Truncated Polynomial Ring)。

例 5.12 设 $N=3$, $a=2-x+3x^2$,$b=1+2x-x^2$,那么:

$$\begin{aligned} a+b&=3+x+2x^2\\ a\otimes b&=2+3x-x^2+7x^3-3x^4\\ &=2+3x-x^2+7-3x\\ &=9-x^2 \end{aligned}$$

截尾多项式环也可以用多项式商环来定义:

$$R=z[x]/(x^N-1)$$

显然,二者是一致的。

定义 5.3 设 $F=[a_0,a_1,\cdots,a_{N-1}]\in R$,$q\in Z$,则:

$$F \bmod q=\sum_{i=0}^{N-1}(a_i \bmod q)x^i$$

定义 5.4 设 $F\in R$,$q\in Z$,如果存在 $G\in R$,使得:

$$F\otimes G=1(\bmod q)$$

称 G 与 F 互逆,G 是 F 的逆,F 也是 G 的逆。

例 5.13 $N=7$, $q=11$,$f(x)=3+2x^2-3x^4+x^6$,$f(x)$在 R 中模 11 的逆为:

$$f_{11}^{-1}(x)=-2+4x+2x^2+4x^3-4x^4+2x^5-2x^6$$

这是因为:

$$f(x)f_{11}^{-1}(x)=-10+22x+22x^3-22x^6(\text{mod }11)=1$$

NTRU 公钥密码算法：设$(R,+,\otimes)$是 $N-1$ 次截尾多项式环，首先选取的参数是大模数 q 和小模数 p。

需要说明的是有限域 GF(p)上的元素取自区间$(-p/2,p/2]$，而不是在区间$[0,p)$上。(mod p)的运算结果也必须限制在区间$(-p/2,p/2]$上，而不是在区间$[0,p)$上，同样(mod q)的运算结果也限制在区间$(-q/2,q/2]$上。这种做法和前面章节中的做法明显不同。

p 一般为奇素数，p 和 q 要求互素。

然后选取两个多项式 f 和 g，要求 f 在环 R 上有模 p 和模 q 的逆元，并定义：

f_p：是多项式 f 在环 R 上模 p 时的逆元，即：

$$f_p\otimes f=1(\text{mod }p)$$

f_q：是多项式 f 在环 R 上模 q 时的逆元，即：

$$f_q\otimes f=1(\text{mod }q)$$

确定公钥是：

$$h=pf_q\otimes g(\text{mod }q)$$

私钥是 f 和 g。

明文 $m\in R$，m 的系数取自区间$(-p/2,p/2]$。

加密算法：

取 $r\in R$，r 的系数取自区间$(-p/2,p/2]$，计算：

$$e=r\otimes h+m(\text{mod }q)$$

e 就是密文。

解密算法：

(1) 计算：

$$a=f\otimes e\text{ mod }q$$

其中，a 的系数取自区间$(-q/2,q/2]$。

(2) 计算：

$$b=a\text{ mod }p$$

其中，b 的系数取自区间$(-p/2,p/2]$。

(3) 计算：

$$c=f_p^{-1}\otimes b\text{ mod }p$$

其中，c 的系数取自区间$(-p/2,p/2]$。于是可得：

$$c=m$$

例 5.14 设 $N=11$，$p=3$，$q=32$，显然 p 是素数，且 p 与 q 互素。

取：

$$f=-1+x+x^2-x^4+x^6+x^9-x^{10}$$
$$g=-1+x^2+x^3+x^5-x^8-x^{10}$$

根据欧几里德扩展算法可求出：

$$f_p^{-1}(x)=1+2x+2x^3+2x^4+x^5+2x^7+x^8+2x^9$$

$$f_q^{-1}=5+9x+6x^2+16x^3+4x^4+15x^5+16x^6+22x^7+20x^8+18x^9+30x^{10}$$

公钥：

$$h = pf_q \otimes g \pmod q$$
$$= 8 + 25x + 22x^2 + 20x^3 + 12x^4 + 24x^5 + 15x^6 + 19x^7 + 12x^8 + 19x^9 + 16x^{10}$$

已知明文：

$$m = -1 + x^3 - x^4 - x^8 + x^9 + x^{10}$$

加密时，先取：

$$r = -1 + x^2 + x^3 + x^4 - x^5 - x^7$$

然后计算出密文：

$$e = r \otimes h + m \pmod{32}$$
$$= 14 + 11x + 26x^2 + 24x^3 + 14x^4 + 16x^5 + 30x^6 + 7x^7 + 25x^8 + 6x^9 + 19x^{10}$$

解密时，先计算：

$$a = f \otimes e \pmod{32}$$
$$= 3 - 7x - 10x^2 - 11x^3 + 10x^4 + 7x^5 + 6x^6 + 7x^7 + 5x^8 - 3x^9 - 7x^{10}$$

然后计算：

$$b = a \bmod 3$$
$$= -x - x^2 + x^3 + x^4 + x^5 + x^7 - x^8 - x^{10}$$

最后计算出明文：

$$c = f_p^{-1} \otimes b \bmod 3$$
$$= (1 + 2x + 2x^3 + 2x^4 + x^5 + 2x^7 + x^8 + 2x^9)$$
$$\otimes (-x - x^2 + x^3 + x^4 + x^5 + x^7 - x^8 - x^{10}) \bmod 3$$
$$= -1 + x^3 - x^4 - x^8 + x^9 + x^{10}$$

解密成功。

根据以上步骤不难得到 NTRU 公钥算法的工作原理：

$$a = f \otimes e \pmod q$$
$$= f \otimes (f \otimes h + m) \pmod q$$
$$= f \otimes (r \otimes pf_q \otimes g + m) \pmod q$$
$$= pr \otimes g + f \otimes m \pmod q$$

于是：

$$f_p \otimes a \pmod p = f_p \otimes pr \otimes g + f_p \otimes f \otimes m \pmod p$$
$$= m \pmod p$$

5.4　概率公钥密码系统

S. Goldwasser 和 S. Micali 于 1984 年首次提出了概率公钥密码体制的概念。不同于其他的公钥密码体制，Goldwasser-Micali 概率公钥密码体制是一种一次一密的公开密钥体制，具有极好的安全性。尽管该体制并不实用，但一经提出便倍受密码学界的关注。

1986 年，M. Blum 和 S. Goldwasser 在此基础上，又提出了 Blum-Goldwasser 概率公钥密码体制，使概率公钥密码体制的实用性显著增强。

Blum-Goldwasser 概率公钥密码体制的基本原理：首先利用 BBS 随机序列生成器产生

密钥流，然后采用流密码技术对明文加密，最后使用公钥密码方法，并结合中国剩余定理传递 BBS 的密钥种子。

下面先介绍 BBS 随机序列生成器的工作原理。

Blum-Blum-Shub（BBS）随机序列发生器：设 $n=pq$，p、q 是两个 $k/2$ 位的素数，满足：

$$p = q = 3 \bmod 4$$

记 $QR(n)$ 表示 n 的平方剩余，种子 s_0 是 $QR(n)$ 中的一个元素。

对于 $1\leqslant i\leqslant t$，定义：

$$s_{i+1} = s_i^2 \bmod n$$

产生的随机序列为：

$$f(s_0) = (z_1, z_2, \cdots, z_t)$$

其中，$z_i = s_i \bmod 2$。

f 称为 Blum-Blum-Shub 随机序列发生器，简称 BBS 随机序列发生器。

例 5.15　设 $n=503\times 607=305\ 321$，$s_0=123\ 456^2 \bmod 305\ 321=64\ 937$，以下是 BBS 随机序列的产生过程：

$$s_1 = 25\ 638 \quad z_1 = 0$$
$$s_2 = 256\ 252 \quad z_2 = 0$$
$$s_3 = 5355 \quad z_3 = 1$$
$$s_4 = 281\ 172 \quad z_4 = 0$$
$$s_5 = 11\ 091 \quad z_5 = 1$$
$$s_6 = 271\ 239 \quad z_6 = 1$$
$$s_7 = 141\ 640 \quad z_7 = 0$$
$$s_8 = 162\ 653 \quad z_8 = 1$$
$$s_9 = 239\ 080 \quad z_9 = 0$$
$$s_{10} = 101\ 990 \quad z_{10} = 0$$
$$s_{11} = 284\ 272 \quad z_{11} = 0$$
$$s_{12} = 39\ 630 \quad z_{12} = 0$$
$$s_{13} = 270\ 997 \quad z_{13} = 1$$
$$s_{14} = 208\ 558 \quad z_{14} = 0$$
$$s_{15} = 104\ 383 \quad z_{15} = 1$$
$$s_{16} = 125\ 483 \quad z_{16} = 1$$
$$s_{17} = 273\ 998 \quad z_{17} = 0$$
$$s_{18} = 133\ 956 \quad z_{18} = 0$$
$$s_{19} = 189\ 445 \quad z_{19} = 1$$

于是，由 BBS 随机序列发生器产生的随机序列是：

$$f(s_0) = \text{“0010110100001011001”}$$

Blum-Goldwasser 概率公钥系统：设 $n=pq$，p、q 满足：

$$p = q = 3 \bmod 4$$

明文空间：　$P=Z_2^m$

密文空间： $C=Z_2^m\times Z_n^*$

密钥： $K=\{(n,p,q):n=pq\}$

其中，n 为公钥，p 和 q 为私钥。

明文 x 用$(x_1,x_2,\cdots,x_m)$表示。

加密算法：

利用种子 $r\in Z_n^*$，通过 BBS 随机序列发生器产生随机序列 z_1、z_2、…、z_m。

计算：

$$s_{m+1}=s_0^{2^{m+1}}\bmod n$$
$$z_i=s_i\bmod 2$$

对于 $1\leqslant i\leqslant m$，计算：

$$y_i=(x_i+z_i)\bmod 2$$

序列 $y=(y_1,y_2,\cdots,y_m,s_{m+1})$就是密文。

解密算法：

计算：

$$a_1=\left(\frac{p+1}{4}\right)^{m+1}\bmod(p-1)$$
$$a_2=\left(\frac{q+1}{4}\right)^{m+1}\bmod(q-1)$$

然后计算：

$$b_1=s_{m+1}^{a_1}\bmod p$$
$$b_2=s_{m+1}^{a_2}\bmod q$$

根据中国剩余定理计算出 s_0，使得：

$$\begin{cases}s_0=b_1\bmod p\\ s_0=b_2\bmod q\end{cases}$$

由 $r=s_0$通过 BBS 随机序列发生器产生序列 z_1、z_2、…、z_m。

对于 $1\leqslant i\leqslant m$ 计算：

$$x_i=(y_i+z_i)\bmod 2$$

于是得到明文 $x=(x_1,x_2,\cdots,x_m)$。

例 5.16 设 $n=719\times 839=603\,241$，$s_0=545\,601^2\bmod 603\,241=321\,413$，$t=20$ 类似于例 5.15 的随机序列产生方法，产生随机序列：

$$z=\text{“1110 1001 0110 1000 1100”}$$

已知明文：

$$x=\text{“1001 1010 1101 0010 0011”}$$

于是得到密文：

$$y=\text{“0111 0011 1011 1010 1111”}$$

并且得到：

$$s_{21}=463\,488$$

解密时，先计算：

$$(p+1)/4=(719+1)/4=180$$

$$(q+1)/4=(839+1)/4=210$$

可得：

$$\begin{aligned}a_1&=((p+1)/4)^{21} \bmod (p-1)\\&=180^{21} \bmod 718\\&=510\\a_2&=((q+1)/4)^{21} \bmod (q-1)\\&=210^{21} \bmod 838\\&=272\end{aligned}$$

接下来

$$\begin{aligned}b_1&=s_{21}^{a_1} \bmod p\\&=463\ 488^{510} \bmod 719\\&=20\\b_2&=s_{21}^{a_2} \bmod q\\&=463\ 488^{272} \bmod 839\\&=76\end{aligned}$$

求解同余方程组：

$$r\equiv 20 \bmod 719$$
$$r\equiv 76 \bmod 839$$

可得：

$$r=321\ 413$$

有了种子 r 后，通过 BBS 随机序列生成器生成：

$$z=\text{“1110 1001 0110 1000 1100”}$$

再与密文：

$$y=\text{“0111 0011 1011 1010 1111”}$$

异或，得到明文：

$$x=\text{“1001 1010 1101 0010 0011”}$$

解密成功。

5.5 习　　题

1. 求解问题：如何产生总额为 173.68 元的钱币，要求钱币由 1 分、2 分、5 分、1 角、2 角、5 角、1 元、2 元、5 元、10 元、20 元、50 元、100 元等币种组合而成，但每个币种的钱币只能使用 1 次。

2. 求解背包问题：$\sum_{i=1}^{n} x_i s_i = z$，其中 $n=9$，$z=586$。

$$s=(s_1,s_2,\cdots,s_9)=(2,4,8,16,32,64,128,256,512)$$

3. 用上题的背包问题构造一个背包公钥系统。

4. 使用 MH 背包公钥密码算法对信息 $m=110101010$ 进行加密，MH 背包公钥体制的

参数是 $p=811$，$a=53$，$t=(53,106,212,424,37,74,148,296,592)$，$s=(1,2,4,8,16,32,64,128,256)$。

5. 使用 MH 背包公钥密码算法对密文 $c=1082$ 进行解密，MH 背包公钥体制的参数是 $p=919$，$a=31$，$t=(31,62,124,248,496,73,146,292,584)$，$s=(1,2,4,8,16,32,64,128,256)$。

6. 已知 $m=100101110111$，求该消息的 Hamming 重量。

7. 已知 $m_1=100101110111$，$m_2=111110101100$，求 m_1 与 m_2 的 Hamming 距离。

8. 求 0101 按偶校验配置的汉明码。

9. 说明按偶校验配置的汉明码 0101101 的纠错过程。

10. 试写出剩余类环 Z_{21} 中的所有零因子和可逆元。

11. 设 R 是有限可交换环且含有单位元，证明：R 中的非零元不是可逆元就是零因子。

12. 设 α 满足 $GF(2^8)$ 上生成多项式 $x^6+x+1=0$，可得 α^7 和 α^{11}。

13. 已知 $N=7$，$p=32$，$q=11$，$f(x)=1+2x^2-3x^4+3x^6$，求 $f(x)$ 在 R 中模 p 和 q 的逆。

14. 设 $n=503\times607=305\ 321$，$s_0=2468^2 \bmod n=289\ 925$，求 BBS 伪随机序列。

第6章　数字签名

日常生活中的很多场合需要签名，如批复文件、签订合同、信用卡消费等，手写签名是一种有效的证明签发者身份的手段。

传统的手写签名通常具备这样一些性质：

(1) 其他人很难伪造签名；

(2) 签名能够被验证；

(3) 签名者事后不能否认自己的签名。

随着密码学的发展，尤其是公钥密码体制的发展，研究者发现：签名的这些性质完全可以通过数字方式来实现。带有数字签名的消息在网络上传播，标识着签名者的身份，服务于电子商务、电子政务和网上银行等系统。

6.1　数字签名方案

虽然签名的目的相同，但数字签名和传统签名还是有一些不同之处。

传统签名是消息的一部分，被签名的文件和签名是不应分割的，而数字签名与消息是独立的，可以将数字签名与消息绑定在一起发送或传输，也可以分开发送或传输。

传统签名的验证往往需要与已知的另一份签名进行比较来实现，而数字签名的验证仅通过验证算法就可以实现。正是因为传统签名的验证需要一些历史信息，所以能够对某一个传统签名进行验证的验证者往往需要较强的专业技能，因此验证者的群体是一个很小的群体；而数字签名的验证者范围要大得多，只要能实现验证算法就能够实现对数字签名的验证。

传统签名是不容易复制的，因为对传统签名的简单复制与签名原件总有区别；而数字签名的复制不受限制，而且还可以很方便地存储、复制和分发。

下面先从数字签名方案的基本概念谈起。

6.1.1　数字签名方案的定义

数字签名方案由$\{P,A,K,S,V\}$组成，并满足：

(1) P 为可能消息的有限集。

(2) A 为可能签名的有限集。

(3) K 为可能密钥的有限集。

(4) 对于每一个 $k\in K$，存在签名算法 $sig_k\in S$ 和验证算法 $ver_k\in V$：

$$sig_k : P \to A$$

$$ver_k(x,y) : P\times A \to \{\text{true},\text{false}\}$$

$$ver_k(x,y)=\begin{cases}\text{true}, & y=sig(x)\\ \text{false}, & y\neq sig(x)\end{cases}$$

显然,对称密码体制下的加密算法就是一个签名算法,加密算法或解密算法可用作相应的验证算法。对称密码体制一般仅用于协议双方之间的签名验证。

假设消息(即明文)为 x,A、B 之间的共享密钥是 k,将消息 x 加密得到密文 y:

$$y=sig_k(x)=E_k(x)$$

将 y 作为消息 x 的签名,验证时,可用加密算法:

$$E_k(x)=y$$

或解密算法:

$$D_k(y)=x$$

来验证签名的有效性:

$$ver(x,y)=\text{true}\Leftrightarrow E_k(x)=y\Leftrightarrow D_k(y)=x$$

当然,这里的签名算法可以是对称密码体制,也可以是公钥密码体制。

考虑到维护密钥通道的开销较大,相对来说,公钥密码体制更适合用作数字签名。但是,与对称密码体制用作数字签名不同,公钥密码体制用私钥进行签名,用公钥进行验证。

6.1.2　RSA 签名方案

下面先介绍一种简单的基于 RSA 的公钥数字签名方案。

RSA 签名方案:令 $n=pq$,p 和 q 是素数,$P=A=Z_n$,定义:

$$K=\{(n,p,q,a,b):ab\equiv 1(\text{mod }\varphi(n))\}$$

其中,n、b 是公钥,p、q、a 是私钥,签名算法和验证算法如下:

$$sig_k(x)=x^a(\text{mod }n)$$

$$ver_k(x,y)=\text{true}\Leftrightarrow x\equiv y^b(\text{mod }n)(x,y\in Z_n)$$

RSA 的签名方案和 RSA 的密码算法非常相似,不同的是签名算法用的是私钥,验证算法用的是公钥,这样能够保证持有公钥的所有参与者都能够对签名进行验证。

例 6.1　在 RSA 签名方案中,取 $p=251$,$q=503$,则:

$$n=251\times 503=126\ 253$$

$$\varphi(n)=250\times 502=125\ 500$$

取 $b=2957$,则:

$$a=2957^{-1}\text{mod }125\ 500=5093$$

其中,n、b 是公钥,p、q、a 是私钥,对 $x=5601$ 进行签名:

$$sig_k(x)=5601^{5093}\text{mod }126\ 253=5093$$

对(5601,5093)进行验证:

$$ver_k(x,y)=\text{true}\Leftrightarrow 5601=5093^{2957}\text{mod }126\ 253$$

于是,(5601,5093)是有效的 RSA 签名。

接下来,对另一组签名(2005,2006)进行验证。由于:

$$ver_k(x,y)=\text{false}\Leftrightarrow 2006^{2957}\text{mod }126\ 253=23\ 717\neq 2005$$

这说明(2005,2006)不是有效的 RSA 签名。

公钥是公开的，因此如果获得公钥就可以验证签名。只要没有私钥，就无法伪造出有效签名。通过公钥去破解私钥，等同与大数因式分解难题，所以 RSA 签名的安全性同样是建立在大数因式分解难题基础之上。

6.2　ElGamal 签名方案与 DSA

通过其他的 NP 完全问题同样可以构造出签名方案，下面的 ElGamal 签名方案就是基于离散对数问题的签名方案。

ElGamal 签名方案：设 p 是素数，$\alpha \in Z_p^*$ 为生成元，$P=Z_p^*$，$A=Z_p^* \times Z_{p-1}$。

密钥空间：

$$K = \{(p,\alpha,\beta,a):\beta = \alpha^a (\bmod p)\}$$

其中，p、α、β 是公钥，a 是私钥，对于随机选取的 $t \in Z_{p-1}^*$，设计签名算法为：

$$sig_k(x,t) = (\gamma,\delta)$$

其中：

$$\gamma = \alpha^t \bmod p$$

$$\delta = (x - a\gamma)t^{-1} \bmod (p-1)$$

验证算法为：

$$ver(x,\gamma,\delta) = \text{true} \Leftrightarrow \beta^\gamma \gamma^\delta \equiv \alpha^x (\bmod p)$$

其中，$x,\gamma \in Z_p^*$，$\delta \in Z_{p-1}$。

验证过程如下：

$$\begin{aligned} & \beta^\gamma \gamma^\delta (\bmod p) \\ = & (\alpha^a)^\gamma (\alpha^t)^\delta (\bmod p) \\ = & \alpha^{a\gamma + t\delta} (\bmod p) (\text{将 } \delta = (x - a\gamma)t^{-1} \bmod (p-1) \text{ 代入}) \\ = & \alpha^x (\bmod p) \end{aligned}$$

例 6.2　在 ElGamal 签名方案中，设 $p=1999$，$\alpha=13$，$\beta=103$，$a=200$，$k=\{1999,13,103,200\}$，需要签名的消息为：

$$x = 999$$

不妨取 $t=191$，则：

$$\gamma = 13^{191} \bmod 1999 = 1476$$

$$\delta = (999 - 200 \times 1476) \times 191^{-1} \bmod 1998 = 1305$$

ElGamal 签名为：

$$sig_k(x,t) = (1476,1305)$$

对签名进行验证：

$$ver(999,1476,1305) = \text{true}$$

这是因为：

$$103^{1476} \times 1476^{1305} \equiv 13^{999} (\bmod 1999)$$

如果有错误签名：

$$sig'_k(x,t) = (1477,1306)$$

对签名进行验证,可得:

$$ver(999,1477,1306) = \text{false}$$

这是因为:

$$103^{1477} \times 1477^{1306} = 1737 \neq 13^{999} (\text{mod } 1999)$$

离散对数问题的难解保证了 ElGamal 的安全性,但也存在着伪造签名的方法。

下面的方法能够绕过求解离散对数问题,而直接构造出一组有效签名。

设 $0 \leqslant i \leqslant p-2, 0 \leqslant j \leqslant p-2$ 且满足:

$$\gcd(j, p-1) = 1$$

通过以下办法计算 γ、δ、x:

$$\gamma = \alpha^i \beta^j \text{mod } p$$
$$\delta = -\gamma j^{-1} \text{mod}(p-1)$$
$$x = -\gamma i j^{-1} \text{mod}(p-1)$$

$j^{-1}\text{mod}(p-1)$是存在的,因为 j 与 $p-1$ 互素。

对签名(x,γ,δ)进行验证:

$$\begin{aligned}
&\beta^{\gamma}\gamma^{\delta} \\
&= \beta^{\alpha^i\beta^j} (\alpha^i\beta^j)^{-\alpha^i\beta^j j^{-1}} (\text{mod } p) \\
&= \beta^{\alpha^i\beta^j} \alpha^{-ij^{-1}\alpha^i\beta^j} \beta^{-\alpha^i\beta^j} (\text{mod } p) \\
&= \alpha^{-ij^{-1}\alpha^i\beta^j} (\text{mod } p) \\
&= \alpha^{-\gamma ij^{-1}} (\text{mod } p) \\
&= \alpha^{x} (\text{mod } p)
\end{aligned}$$

说明签名(x,γ,δ)能够通过验证。

例 6.3 ElGamal 签名方案中,设 $p=1999, \alpha=13, \beta=103, a=200, k=\{1999,13,103,200\}$,取 $i=109, j=199$,则:

$$\gamma = 13^{109} \times 103^{199} \text{ mod } 1999 = 1476$$
$$\delta = -1476 \times 199^{-1} \text{mod } 1998 = 1305$$
$$x = -1476 \times 109 \times 199^{-1} \text{mod } 1998 = 999$$

伪造的签名为:

$$sig_k(999) = (1476,1305)$$

对签名进行验证:

$$ver(999,1476,1305) = \text{true}$$

是因为$103^{1476} \times 1476^{1305} \equiv 13^{999} (\text{mod } 1999)$。

另外一种伪造签名基于已知一组有效签名(γ,δ,x),设:

$$0 \leqslant h \leqslant p-2, \quad 0 \leqslant i \leqslant p-2, \quad 0 \leqslant j \leqslant p-2$$

并且满足:

$$\gcd(h\gamma - j\delta, p-1) = 1$$

计算出:

$$\lambda = \gamma^h \alpha^i \beta^j \text{ mod } p \tag{6.1}$$
$$\mu = \delta\lambda (h\gamma - j\delta)^{-1} \text{ mod } (p-1) \tag{6.2}$$

$$x' = \lambda(hx + i\delta)(h\gamma - j\delta)^{-1} \bmod (p-1) \tag{6.3}$$

$(h\gamma - j\delta)^{-1} \bmod (p-1)$是存在的,因为 $h\gamma - j\delta$ 与 $p-1$ 互素。

对签名(λ,μ,x')进行验证,可得:

$$\beta^{\lambda}\lambda^{\mu} = \alpha^{x'} \pmod p \tag{6.4}$$

下面进行简单的证明。

根据式(6.1)可得:

$$\begin{aligned}\lambda &= \alpha^{ht}\alpha^{i}\alpha^{aj} \pmod p \\ &= \alpha^{ht+i+aj} \pmod p\end{aligned} \tag{6.5}$$

将式(6.5)和 $\beta = \alpha^{a} \pmod p$代入式(6.4)得:

$$\alpha^{\lambda a}\alpha^{(ht+i+aj)\mu} = \alpha^{x'} \pmod p$$

等价于:

$$\lambda a + (ht + i + aj)\mu = x' \pmod{p-1}$$

将式(6.2)和式(6.3)代入下式:

$$\lambda a + (ht + i + aj)\mu - x' \pmod{p-1}$$

可得:

$$\begin{aligned}&\lambda a + (ht + i + aj)\mu - x' \pmod{p-1} \\ &= \lambda[a + (ht + i + aj)\delta(h\gamma - j\delta)^{-1} - (hx + i\delta)(h\gamma - j\delta)^{-1}] \pmod{p-1} \\ &= \lambda[a + (ht\delta + aj\delta - hx)(h\gamma - j\delta)^{-1}] \pmod{p-1} \\ &= \lambda\{a + [h(t\delta - x) + aj\delta](h\gamma - j\delta)^{-1}\} \pmod{p-1}\end{aligned}$$

由 $\delta = (x - a\gamma)t^{-1} \bmod (p-1)$得:

$$\begin{aligned}&x - \delta t = a\gamma \pmod p \\ &\lambda a + (ht + i + aj)\mu - x' \pmod{p-1} \\ &= \lambda\{a + [-ah\gamma + aj\delta](h\gamma - j\delta)^{-1}\} \pmod{p-1} \\ &= \lambda\{a - a[h\gamma - j\delta](h\gamma - j\delta)^{-1}\} \pmod{p-1} \\ &= 0 \pmod{p-1}\end{aligned}$$

命题得证。

例 6.4　ElGamal 签名方案中:设 $p=4079, \alpha=13, \beta=1621, a=200$。

已知密钥:

$$k = \{4079, 13, 1621, 200\}$$

已知一组有效签名:

$$(\gamma, \delta, x) = (2429, 4005, 999)$$

取:

$$h = 888, \quad i = 129, \quad j = 555$$

满足条件:

$$\begin{aligned}&\gcd(888 \times 2429 - 555 \times 4005, 4078) \\ &= \gcd(3503, 4078) \\ &= 1\end{aligned}$$

则有:

$$\lambda = 2429^{888} \times 13^{129} \times 1621^{555} \bmod 4079 = 2940$$

$$\begin{aligned}\mu &= 4005 \times 2940 \times (888 \times 2429 - 555 \times 4005)^{-1} \bmod 4078 \\ &= 4005 \times 2940 \times 1539 \bmod 4078 \\ &= 1508\end{aligned}$$

$$\begin{aligned}x' &= 2940 \times (888 \times 999 + 129 \times 4005)(888 \times 2429 - 555 \times 4005)^{-1} \bmod 4078 \\ &= 2940 \times 925 \times 1539 \bmod 4078 \\ &= 2008\end{aligned}$$

伪造的签名为：

$$sig_k(2008) = (2940,1508)$$

对签名进行验证：

$$ver(2008,2940,1508) = \text{true}$$

这是因为：

$$1621^{2940} \times 2940^{1508} (\bmod 4079) = 2343$$

$$13^{999} (\bmod 4079) = 2343$$

虽然以上的两种伪造 ElGamal 签名都能通过验证，但是这两种伪造并不会对 ElGamal 签名算法本身造成实质的威胁，因为这两种伪造都只能产生特殊的签名，而不能对任意的 x 计算出签名。

相对而言，对 ElGamal 数字签名方案的不当使用却是更危险的情形。

例如，t 作为方案中的一个重要参数不能暴露。如果 t 暴露了，那么由

$$\delta = (x - a\gamma)t^{-1} \bmod (p-1)$$

可得：

$$a = (x - t\delta)\gamma^{-1} \bmod (p-1)$$

于是 a 被求出，a 的求出将导致整个 ElGamal 签名算法被破解。

例 6.5　ElGamal 签名方案中，设 $p=2039$，$\alpha=13$，$\beta=1428$，p、α、β 是公开的，$a=211$ 是保密的。

消息为：

$$x = 819$$

签名为：

$$(\gamma,\delta) = (647,420)$$

如果攻击者知道 $t=167$，那么就可以算出 a 的值：

$$\begin{aligned}a &= (819 - 167 \times 420) \times 647^{-1} \bmod (2039-1) \\ &= 2009 \times 63 \bmod 2038 \\ &= 211\end{aligned}$$

又如，在两次数字签名中使用相同的 t 也是很危险的。如果出现这种情况，那么设：

$$\beta^{\gamma}\gamma^{\delta_1} \equiv \alpha^{x_1} (\bmod p)$$

$$\beta^{\gamma}\gamma^{\delta_2} \equiv \alpha^{x_2} (\bmod p)$$

于是：

$$\alpha^{x_1 - x_2} \equiv \alpha^{t(\delta_2 - \delta_1)} (\bmod p)$$

$$x_1 - x_2 \equiv t(\delta_2 - \delta_1)(\bmod p-1) \tag{6.6}$$

不妨设 $d=\gcd(\delta_2-\delta_1, p-1)$，显然 d 同时满足：

$$d \mid (p-1)$$
$$d \mid (\delta_2-\delta_1)$$
$$d \mid (x_1-x_2)$$

令：

$$x' = (x_1-x_2)/d$$
$$\delta' = (\delta_2-\delta_1)/d$$
$$p' = (p-1)/d$$

式(6.5)可转化为：

$$x' \equiv t\delta' (\bmod\ p')$$
$$t = x'(\delta')^{-1} (\bmod\ p')$$

$(\delta')^{-1}(\bmod\ p')$是存在的，因为 $\gcd(\delta', p')=1$。

于是：

$$t = x'(\delta')^{-1} + ip' (\bmod\ p)$$

其中，$0\leqslant i\leqslant d-1$。虽然这样求出的 t 是 d 组可能的结果，但可以通过验证：

$$\gamma = \alpha^t \bmod p$$

来得到最终的 t 值。

例 6.6　ElGamal 签名方案中：设 $p=1009$，$\alpha=13$，$\beta=635$，a 未知，已知两个签名分别为：

$$(\gamma,\delta_1,x_1) = (388,833,999)$$
$$(\gamma,\delta_2,x_2) = (388,323,501)$$

即：

$$635^{388} \times 388^{833} = 13^{999} \bmod 1009$$
$$635^{388} \times 388^{323} = 13^{501} \bmod 1009$$

于是：

$$13^{999-501} = 13^{t(323-833)} \bmod 1009$$
$$498 = 498t \bmod 1008$$

不妨设：

$$d = \gcd(498,1008) = 6$$

令：

$$x' = 498/6 = 83$$
$$\delta' = 498/6 = 83$$
$$p' = 1008/6 = 168$$

于是：

$$83 = 83t \bmod 168$$
$$t_0 = 83 \times 83^{-1} \bmod 168 = 1$$

t 的表达式为：

$$t = t_0 + ip' \bmod 1009 = 1 + 83i \bmod 1009$$

t 的候选结果为：

$$t_1 = 1 + 83 \times 1 \bmod 1009 = 84 \bmod 1009 = 84$$

$$t_2 = 1 + 83 \times 2 \bmod 1009 = 167 \bmod 1009 = 167$$
$$t_3 = 1 + 83 \times 3 \bmod 1009 = 250 \bmod 1009 = 250$$
$$t_4 = 1 + 83 \times 4 \bmod 1009 = 333 \bmod 1009 = 333$$
$$t_5 = 1 + 83 \times 5 \bmod 1009 = 416 \bmod 1009 = 416$$
$$t_6 = 1 + 83 \times 6 \bmod 1009 = 449 \bmod 1009 = 449$$

在 6 个候选的 t 中进行筛选，其中 167 符合要求，因为：

$$13^{167} = 388 \bmod 1009$$

美国 NIST 于 1991 年 8 月提出了数字签名标准(Digital Signature Standard，DSS)，该标准中的数字签名算法 DSA(Digital Signature Algorithm)由 D. W. Kravitz 设计。

DSA 是 Schnorr 和 ElGamal 签名算法的变种，同样也是建立在离散对数问题之上的。

DSA 签名算法：设 p 是 L 位的素数(L 介于 512 和 1024 之间，同时又是 64 的倍数)，Z_p 上的离散对数问题是一个难解问题，q 是一个 160 位的素数，且满足：

$$q \mid (p-1)$$

取 q 阶元 $\alpha \in Z_p^*$，满足：

$$\alpha^q \equiv 1 \bmod p$$

消息空间和签名空间分别为：

$$P = Z_p^*, \quad A = Z_q^* \times Z_q^*$$

定义密钥：

$$K = \{(p, q, \alpha, \beta, a) : \beta = \alpha^a (\bmod p)\}$$

其中，p、q、α、β 是公钥，a 是私钥，$0 \leqslant a \leqslant q-1$。

取随机数 t，$0 \leqslant t \leqslant q-1$。

签名算法：

$$sig_k(x, t) = (\gamma, \delta)$$

其中：

$$\gamma = (\alpha^t \bmod p) \bmod q$$
$$\delta = (\text{SHA}(x) + a\gamma)t^{-1} \bmod q$$

验证算法：

$$ver(x, \gamma, \delta) = \text{true} \Leftrightarrow (\alpha^{e_1}\beta^{e_2} \bmod p) \bmod q = \gamma$$

其中：

$$e_1 = \text{SHA}(x)\delta^{-1} \bmod q$$
$$e_2 = \gamma\delta^{-1} \bmod q$$

算法中的消息 x 一般以二进制串的方式编码。SHA 是 SHA-1 算法，SHA-1 算法是一种 Hash 函数，关于 Hash 函数将在第 7 章介绍，此处可以把 Hash 函数理解成一种压缩算法。

6.3 ECDSA

基于椭圆曲线上的离散对数问题也能构造出高效的数字签名算法，而且近年来椭圆曲线数字签名算法 ECDSA 逐渐成为主流。

椭圆曲线数字签名算法 ECDSA：已知椭圆曲线

$$y^2 \equiv x^3 + ax + b \pmod p$$

定义在 Z_p 上($p>3$)，随机选择整数 $d(1\leqslant d\leqslant n-1)$，设 G 是生成元。

$$Q = G^d$$

其中，Q 是公钥，d 是私钥。设消息为 m，ECDSA 签名算法如下：

(1) 随机选择整数 k，$1\leqslant k\leqslant n-1$。

(2) 计算出：

$$G^k = (x_1,\ y_1)$$

(3) 令：

$$r = x_1 \bmod n$$

(4) 计算出 m 的消息摘要 SHA-1(m)并将它转化成一个整数 e。数字摘要将在 7.1 节中介绍，可以先将 SHA-1(m)理解成一个普通函数，它将消息 m 变换成一个有限长度的串。

(5) 计算：

$$s = k^{-1}(e + dr) \bmod n$$

(6) (r,s)就是消息 m 的签名。

对于消息 m 和签名(r,s)，相应的验证算法如下：

(1) 先验证 r 和 s 是否满足：

$$1 \leqslant r \leqslant n-1,\quad 1 \leqslant s \leqslant n-1$$

(2) 计算 m 的消息摘要 SHA-1(m)，并将它转化成一个证书 e。

(3) 计算出：

$$w = s^{-1} \bmod n$$

$$u_1 = ew \bmod n$$

$$u_2 = rw \bmod n$$

(4) 设：

$$X = G^{u_1}Q^{u_2} = (x_1,\ y_1)$$

(5) 计算：

$$v = x_1 \bmod n$$

(6) 得出结论：

$$ver(m,r,s) = \text{true} \Leftrightarrow v = r$$

下面对 ECDSA 算法进行简单的验证：

因为：

$$s = k^{-1}(e + dr) \bmod n$$

所以：

$$\begin{aligned} k &= s^{-1}(e + dr) \bmod n \\ &= s^{-1}e + s^{-1}dr \bmod n \\ &= we + wdr \bmod n \\ &= (u_1 + u_2 d) \bmod n \end{aligned}$$

于是根据椭圆曲线的运算定义有：

$$G^{u_1}Q^{u_2} = G^{u_1}G^{du_2} = G^{u_1+du_2} = G^k$$

如果 r 和 s 是真实签名，则：

$$v = r = x_1 \bmod n$$

其中，x_1 满足：

$$X = (x_1, y_1) = G^{u_1} Q^{u_2} = G^k$$

反之亦然，证毕。

例 6.7 定义在有限域上的椭圆曲线：

$$E: y^2 = x^3 + 8x + 10 \bmod 23$$

取生成元 $G=(7,8)$，它的阶为 $n=21$，取整数 $d=11$，计算公钥：

$$Q = (7,8)^{11} = (16,5)$$

将 m 的消息摘要 SHA-1(m)转化成一个整数(忽略数字摘要的过程)，不妨设

$$e = 20$$

实施 ECDSA 签名算法：

$$k = 5$$
$$G^k = (7,8)^5 = (19,12)$$
$$r = 19$$
$$s = k^{-1}(e + dr) = 5^{-1}(20 + 11 \times 19) \bmod 21 = 8$$

得到消息 m 的签名(19,8)，下面对消息 m 和签名(19,8)进行验证：

$$w = 8^{-1} \bmod 21 = 8$$
$$u_1 = 20 \times 8 \bmod 21 = 13$$
$$u_2 = 19 \times 8 \bmod 21 = 5$$
$$X = G^{u_1} Q^{u_2} = (7,8)^{13}(16,5)^5 = (19,12)$$
$$v = 19$$

因为 $v=r$，所以该签名是有效签名。对非法签名(17,13)进行验证：

$$w = 13^{-1} \bmod 21 = 13$$
$$u_1 = 20 \times 13 \bmod 21 = 8$$
$$u_2 = 17 \times 13 \bmod 21 = 11$$
$$X = G^{u_1} Q^{u_2} = (7,8)^8(16,5)^{11} = (22,22)$$
$$v = 22$$

因为 $v \neq r$，所以签名(17,13)不是消息 m 的有效签名。

6.4 一次签名方案

与前面的常规签名方案相比，一次签名是一类特殊的签名。这一类签名对于一条消息签名时，具有很高的安全性，但是对于多条消息签名时就不安全了。

之所以称一次签名方案是一类签名，是因为一次签名可以通过任意的单向函数来构造。下面的 Lamport 签名是 1979 年提出的最早的一次签名方案。

Lamport 签名方案： k 是整数，$P=\{0,1\}^k$，$f: Y \to Z$ 是单向函数，令 $A=Y^k$，$y_{i,j} \in Y$ 随

机选取$(1\leqslant i\leqslant k, j=0,1)$，设 $z_{i,j}\in f(y_{i,j})(1\leqslant i\leqslant k, j=0,1)$，密钥 K 包括 $2k$ 个 y 和 $2k$ 个 z。其中，y 是私钥，z 是公钥。

签名算法：

$$sig_K(x_1,\cdots,x_k)=(y_{1,x_1},\cdots,y_{k,x_k})$$

验证算法：

$$ver_K(x_1,\cdots,x_k,a_1,\cdots,a_k)=\text{true}\Leftrightarrow f(a_i)=z_{i,x_i}$$

其中，$1\leqslant i\leqslant k$。

例 6.8 Lamport 签名方案中：$p=7177$，$\alpha=7$ 是生成元。

选取 6 个随机数构成私钥 y：

$$y=\begin{Bmatrix}2153 & 1245\\ 283 & 3421\\ 5148 & 3237\end{Bmatrix}$$

公钥 z 通过定义的单向函数：

$$f(x)=7^x \bmod 7177$$

求得：

$$z=\begin{Bmatrix}4613 & 3819\\ 4565 & 2881\\ 1499 & 5704\end{Bmatrix}$$

密钥：

$$Key=\{p,\alpha,y,z\}$$

其中，y 是密钥，z 是公钥。

需要签名的消息是：

$$x=(1,0,1)$$

实施 Lamport 名：

$$sig(x)=(1245,283,3237)$$

对 Lamport 签名进行验证：

$$ver(x)=\text{true}\Leftrightarrow\begin{cases}7^{1245}\bmod 7177=3819\\ 7^{283}\bmod 7177=4565\\ 7^{3237}\bmod 7177=5704\end{cases}$$

Lamport 签名方案是非常安全的，攻击者在不知道密钥的情况下是不可能伪造签名的。但是，同一个 Lamport 签名方案不能对两个不同消息进行签名，因为根据两个不同的消息签名，可以得到更多的签名。

例如，用同一个 Lamport 签名方案对消息(1,0,0)和(0,1,0)签名，分别得到(y_{11}, y_{20}, y_{30})，(y_{10}, y_{21}, y_{30})，那么就可以得到(1,1,0)和(0,0,0)的签名(y_{11}, y_{21}, y_{30})，(y_{10}, y_{20}, y_{30})。

6.5 不可抵赖签名方案

只要有签名者的公钥就可以鉴别签名是有效的，还是伪造的。但是，明明是有效的签名，签名者事后却不承认该怎么办？这就是抵赖的问题。

应该说前面的签名是不能实现“不可抵赖”功能的，解决这个问题的办法是在原有的签名和认证方案基础上，增加一个由签名者参与的伪造证明协议。

伪造证明协议中使用一种挑战/响应(Challenge & Response)方式。这里的挑战/响应方式是指验证方先提出一个挑战数据，由签名方进行响应，然后由验证方根据这个响应数据来得出某个结论。

在不可抵赖问题上，有以下情况需要考虑：

(1) 签名者有可能否认早期的有效签名；

(2) 签名者有权拒绝参与挑战/响应方式的验证；

(3) 签名者虽然参与了挑战/响应方式的验证，但并不配合或总是提供假数据。

伪造证明协议需要签名方参与，用来证明一个签名确实是伪造而不是签名方签发的。如果签名方不配合或配合后仍不能证伪，就说明签名方抵赖了。

因此完整的不可抵赖签名方案应包括一个签名算法、一个验证协议和一个伪造证明协议。

Chaum 和 van Antwerpen 于 1989 年率先提出了下面的不可抵赖签名算法。

Chaum-van Antwerpen 签名算法和验证协议：q 是素数，$p=2q+1$ 也是素数，令 $\alpha\in Z_p^*$(阶为 q)，定义：

$$\beta = \alpha^a \bmod p \quad (1 \leqslant a \leqslant q-1)$$

G 表示 Z_p^* 阶为 q 的乘法子群(由模 p 的平方剩余组成)，$P=A=G$，定义：

$$K = \{(p,\alpha,\beta,a):\beta = \alpha^a \bmod p\}$$

签名算法为：

$$y = x^a \bmod p$$

对于 x、$y\in G$，验证使用以下协议：

(1) 验证方随机选取 e_1、$e_2\in Z_q^*$。

(2) 验证方计算：

$$c = y^{e_1}\beta^{e_2} \bmod p$$

然后将 c 发给签名方。

(3) 签名方计算：

$$d = c^{a^{-1} \bmod q} \bmod p$$

然后将 d 发给验证方。

(4) 验证方认可签名，当且仅当：

$$d \equiv x^{e_1}\alpha^{e_2} \bmod p$$

下面简单地对签名进行验证：

根据第(1)、(3)步，可得 $d=c^{a^{-1}}(\bmod p)=y^{e_1a^{-1}}\beta^{e_2a^{-1}}(\bmod p)$

因为 $\beta=\alpha^a$，所以：

$$\alpha = \beta^{a^{-1}}$$

类似的

$$x = y^{a^{-1}}$$

于是：

$$d = x^{e_1}\alpha^{e_2} \pmod p$$

例 6.9　在 Chaum-van Antwerpen 签名方案中，$p=1019$，$q=509$，$p=2q+1$，取 $\alpha=16$，$\beta=817$。

密钥：

$$Key=\{1019,16,817,222\}$$

其中，(1019，16，817)是公钥，222 是私钥。

消息：

$$x = 201$$

实施签名算法：

$$y = 16^{222} = 834$$

验证方选取：

$$e_1 = 48, \quad e_2 = 233$$

并计算出：

$$c = 834^{48} \times 817^{233} \bmod 1019 = 89$$

签名方计算：

$$d = 89^{16^{-1} \bmod 509} \bmod 1019 = 668$$

验证方进行有效性验证：

$$ver = \text{true} \Leftrightarrow 201^{48} \times 46^{233} \bmod 1019 = 668$$

于是，验证方可得出结论：该签名是有效签名。

下面的定理将说明很难使一个非法签名通过验证。

定理 6.1　对于一个伪造的签名 $y \neq x^a \pmod p$，那么验证方能够认可 y 是一个有效签名的概率仅为 $1/q$。

证明：因为 y 和 β 都是阶为 q 的乘法群 G 中的元素，所以每一个可能的 c 恰好对应于 q 个有序对(e_1, e_2)。当签名方接收到 c 时，他无法知道 c 是验证方用这 q 种可能有序对(e_1, e_2)中的具体哪一对构造出来的。

接下来说明，如果 $y \neq x^a \pmod p$，那么 β 所做的任何一个响应可能的回答 $d \in G$ 都恰好与这 q 个有序对(e_1, e_2)中的一个一一对应。

因为 α 是 G 的生成元，所以 G 中每个元素都可以写成 α 的指数形式。

不妨设：

$$c = \alpha^h, \quad d = \alpha^i, \quad x = \alpha^j, \quad y = \alpha^k$$

其中，h、i、j、$k \in Z_q$。

考虑下列两个同余式：

$$c = y^{e_1}\beta^{e_2} \pmod p$$
$$d = x^{e_1}\alpha^{e_2} \pmod p$$

两个同余式等价于：

$$\begin{cases} h = ke_1 + ae_2 \pmod q \\ i = je_1 + e_2 \pmod q \end{cases} \tag{6.7}$$

因为 $y \neq x^a \pmod p$，所以：

$$k \neq aj$$

于是，同余方程组的系数矩阵

$$\begin{bmatrix} k & a \\ j & 1 \end{bmatrix}$$

满秩，方程组(6.7)有唯一解。

也就是说每一个 $d \in G$ 都恰好对应着 q 个有序对(e_1, e_2)中的一个，这也证明了验证方能够以 $1/q$ 的概率认可 y 是一个有效签名。

以上是验证算法，下面是伪造证明协议。

Chaum-van Antwerpen 伪造证明协议：验证方随机选取 e_1、$e_2 \in Z_q^*$，计算出：

$$c_1 = y^{e_1} \beta^{e_2} \bmod p$$

将 c_1 发送给签名方。

(1) 签名方计算：

$$d_1 = c_1^{a^{-1} \bmod q} \bmod p$$

然后，将 d_1 发送给验证方。

(2) 验证方验证是否：

$$d_1 \neq x^{e_1} \alpha^{e_2} \pmod p$$

验证方又随机选取 f_1、$f_2 \in Z_q^*$，计算出：

$$c_2 = y^{e_1} \beta^{e_2} \bmod p$$

将 c_2 发送给签名方。

(3) 签名方计算：

$$d_2 = c_2^{a^{-1} \bmod q} \bmod p$$

然后将 d_2 发送给验证方。

(4) 验证方验证是否：

$$d_2 \neq x^{f_1} \alpha^{f_2} \pmod p$$

(5) 验证方可以确定 y 是伪造的，当且仅当：

$$(d_1 \alpha^{-e_2})^{f_1} \equiv (d_2 \alpha^{-f_2})^{e_1} \pmod p$$

下面进行简单的验证。

因为：

$$d_1 = c_1^{a^{-1}} \pmod p$$

$$c_1 = y^{e_1} \beta^{e_2} \pmod p$$

$$\beta = \alpha^a \pmod p$$

所以：

$$(d_1 \alpha^{-e_2})^{f_1}$$

$$= ((y^{e_1} \beta^{e_2})^{a^{-1}} \alpha^{-e_2})^{f_1} \pmod p$$

$$= y^{e_1 f_1} \beta^{e_2 \alpha^{-1} f_1} \alpha^{-e_2 f_1} \pmod p$$
$$= y^{e_1 f_1} \alpha^{e_2 f_1} \alpha^{-e_2 f_1} \pmod p$$
$$= y^{e_1 f_1} \pmod p$$

同理可得：

$$(d_2 \alpha^{-e_1})^{f_2} = y^{e_1 f_1} \pmod p$$

于是：

$$(d_1 \alpha^{-e_2})^{f_1} = (d_2 \alpha^{-e_1})^{f_2} \pmod p$$

例 6.10 Chaum-van Antwerpen 伪造证明协议中，$p=1019$，$q=509$，$p=2q+1$。

取 $\alpha=16$，$\beta=817$，密钥 $Key=\{1019,16,817,222\}$，其中，(1019,16,817)是公钥，222 是私钥。

消息 $x=286$，现有签名 $y=83$，验证方选取 $e_1=45$，$e_2=237$，计算出：

$$c_1 = 83^{45} \times 817^{237} \bmod 1019 = 244$$

签名方计算：

$$d_1 = 244^{222^{-1} \bmod 509} \bmod 1019 = 629$$

验证方验证：

$$ver1 = \text{false} \Leftrightarrow 286^{45} \times 16^{222} = 770 \neq 629 \pmod{1019}$$

验证方再选取 $f_1=125$，$f_2=9$，并计算出：

$$c_2 = 83^{125} \times 817^{9} \bmod 1019 = 644$$

签名方计算：

$$d_2 = 644^{222^{-1} \bmod 509} \bmod 1019 = 930$$

验证方又验证：

$$ver2 = \text{false} \Leftrightarrow 286^{125} \times 16^{9} = 57 \neq 930 \pmod{1019}$$

验证方再验证：

$$ver3 = \text{true} \Leftrightarrow (629 \times 16^{-237})^{125} = (930 \times 16^{-9})^{45} = 658 \pmod{1019}$$

验证方最终可以得出结论：签名方可以证明此签名不是自己签发的有效签名。也就是说，签名方并没有抵赖。

6.6 Fail-stop 签名方案

Fail-stop 签名满足的是另一种需求：对于一个消息，攻击者可能伪造出一个签名通过验证，虽然概率很小。当签名者发现了这个“有效的签名”有问题时，需要证明这个是伪造的签名。显然，光靠原来的验证算法是不够的，与不可抵赖签名解决方案类似，这时有效签名者需要参与进来形成一种证伪协议，与验证者共同揭穿这个伪造的签名。此外，还要求这个签名方案也是一次签名方案。Fail-stop 签名提供更强大的安全性。

van Heyst & Pedersen Fail-stop 签名方案：q 是素数，$p=2q+1$ 也是素数，令 $\alpha \in Z_p^*$，阶为 q，取 $1 \leqslant a_0 \leqslant q-1$，则：

$$\beta = \alpha^{a_0} \bmod p$$

密钥：

$$\{(p, q, \alpha, \beta \mid a_0)\}$$

其中，p、q、α、β 是公钥，a_0 是私钥。

消息空间和签名空间：

$$P = Z_q, \quad A = Z_q \times Z_q$$

选取：

$$K = (\gamma_1, \gamma_2, a_1, a_2, b_1, b_2)$$

其中：

$$\gamma_1 = \alpha^{a_1} \beta^{a_2} \bmod p$$

$$\gamma_2 = \alpha^{b_1} \beta^{b_2} \bmod p$$

$$a_1, a_2, b_1, b_2 \in Z_q$$

对于 $K=(\gamma_1, \gamma_2, a_1, a_2, b_1, b_2)$ 和 $x \in Z_q$，签名算法为：

$$sig_K(x) = (y_1, y_2)$$

其中：

$$y_1 = a_1 + xb_1 \bmod q$$

$$y_2 = a_2 + xb_2 \bmod q$$

验证算法为：

$$ver_K(x, y) = \text{true} \Leftrightarrow \gamma_1 \gamma_2^x = \alpha^{y_1} \beta^{y_2} \pmod p$$

例 6.11 在 Fail-stop 签名方案中，已知 $p=1019$，$q=509$，$p=2q+1$，取 $\alpha=11$，$\beta=373$，相应的私钥为：

$$a_0 = 23$$

选取：

$$a_1 = 408, \quad a_2 = 302, \quad b_1 = 223, \quad b_2 = 199$$

可计算出：

$$\gamma_1 = 11^{408} \times 373^{302} (\bmod 1019) = 124$$

$$\gamma_2 = 11^{223} \times 373^{199} (\bmod 1019) = 961$$

消息为：

$$x = 200$$

签名为：

$$y = (408 + 200 \times 223, 302 + 200 \times 199) \bmod 509 = (216, 400)$$

对签名进行验证：

$$ver_K(x, y) = \text{true} \Leftrightarrow 124 \times 961^{200} = 11^{216} \times 373^{400} = 565 (\bmod 1019)$$

因此，y 是有效的 Fail-stop 签名。

定义 6.1 如果 $\gamma_1 = \gamma_1'$，$\gamma_2 = \gamma_2'$，称 $\{\gamma_1, \gamma_2, a_1, a_2, b_1, b_2\}$ 与 $\{\gamma_1', \gamma_2', a_1', a_2', b_1', b_2'\}$ 为等价密钥。

定理 6.2 假设 K 和 K' 是等价密钥，如果 $ver_K(x, y) = \text{true}$，那么 $ver_K{}'(x, y) = \text{true}$。

证明：假设 $K = \{\gamma_1, \gamma_2, a_1, a_2, b_1, b_2\}$ 和 $K' = \{\gamma_1', \gamma_2', a_1', a_2', b_1', b_2'\}$ 是两个等价密钥，于是有：

$$\gamma_1 = \alpha^{a_1} \beta^{a_2} (\bmod p) = \alpha^{a_1'} \beta^{a_2'} (\bmod p)$$

$$\gamma_2 = \alpha^{b_1}\beta^{b_2}(\bmod\ p) = \alpha^{b'_1}\beta^{b'_2}(\bmod\ p)$$

假设使用密钥 K 对消息 x 进行签名，签名为：

$$(y_1, y_2)$$

其中：

$$y_1 \equiv a_1 + xb_1(\bmod\ q)$$
$$y_2 \equiv a_2 + xb_2(\bmod\ q)$$

用 K' 来验证签名 (y_1, y_2)：

$$\begin{aligned}&\alpha^{y_1}\beta^{y_2}(\bmod\ p)\\ &= \alpha^{a'_1+xb'_1}\beta^{a'_2+xb'_2}(\bmod\ p)\\ &= (\alpha^{a'_1}\beta^{a'_2})(\alpha^{b'_1}\beta^{b'_2})^x(\bmod\ p)\\ &= \gamma_1\gamma_2^x(\bmod\ p)\end{aligned}$$

命题得证。说明 y 既可以通过 K 来验证，也可以通过 K' 来验证。

定理 6.3 对于 K，有 $y=sig_K(x)$，存在 q 个与 K 等价的密钥使得：

$$y = \mathrm{sig}_{K'}(x)$$

证明：假设 γ_1、γ_2 是 K 的公开部分，先考虑使得以下同余方程组成立的 (a_1, a_2, b_1, b_2) 的个数：

$$\gamma_1 \equiv \alpha^{a_1}\beta^{a_2}(\bmod\ p)$$
$$\gamma_2 \equiv \alpha^{b_1}\beta^{b_2}(\bmod\ p)$$
$$y_1 \equiv a_1 + xb_1(\bmod\ q)$$
$$y_2 \equiv a_2 + xb_2(\bmod\ q)$$

α 是 G 的生成元，所以存在 c_1、c_2 和 a_0 使得：

$$\gamma_1 \equiv \alpha^{c_1}(\bmod\ p)$$
$$\gamma_2 \equiv \alpha^{c_2}(\bmod\ p)$$
$$\beta \equiv \alpha^{a_0}(\bmod\ p)$$

于是可将前面的同余方程组转化为：

$$c_1 \equiv a_1 + a_0a_2(\bmod\ q)$$
$$c_2 \equiv b_1 + b_0b_2(\bmod\ q)$$
$$y_1 \equiv a_1 + xb_1(\bmod\ q)$$
$$y_2 \equiv a_2 + xb_2(\bmod\ q)$$

也可以写成矩阵的形式：

$$\begin{bmatrix}1 & a_0 & 0 & 0\\ 0 & 0 & 1 & a_0\\ 1 & 0 & x & 0\\ 0 & 1 & 0 & x\end{bmatrix}\begin{bmatrix}a_1\\ a_2\\ b_1\\ b_2\end{bmatrix} = \begin{bmatrix}c_1\\ c_2\\ y_1\\ y_2\end{bmatrix}$$

该系数矩阵的秩是 3，因此方程组的解空间的维数为 $4-3=1$，在有限域上恰有 q 个解。

下面不加证明地给出另一个引理：

设 K 是一个密钥，如果 $y=sig_K(x)$，$ver_K(x', y')=\mathrm{true}$，$x'\neq x$，那么最多存在一个 K' 满足：

$$y = \mathrm{sig}_{K'}(x), \quad y' = \mathrm{sig}_{K'}(x')$$

综上所述,可以总结出这样一个结论:对于一个有效的消息签名对(x,y),存在着q个与原K不同的密钥都能够验证该签名。但是,如果换一个消息$x'\neq x$,那么这q个不同的密钥将产生各不相同的签名。基于这个原因,换一个说法就是下面Fail-stop方案的关键定理。

定理 6.4 如果$sig_K(x)=y$且$x'\neq x$,攻击者能够计算$sig_K(x')$的概率是$1/q$。

在Fail-stop签名方案中,q越大,受攻击的概率越小。虽然伪造的可能性比较小,但既然有伪造的可能,就有证明伪造的需求。

假设出现了这种情况:攻击者拥有一对(x',y''),满足$ver(x',y'')=\mathrm{true}$,但$y''\neq sig_K(x')$,即:

$$\gamma_1\gamma_2^{x'} \equiv \alpha^{y_1''}\beta^{y_2''} \pmod p$$

签名方可计算x'的真正签名:

$$\gamma_1\gamma_2^{x'} \equiv \alpha^{y_1'}\beta^{y_2'} \pmod p$$

于是:

$$\alpha^{y_1''}\beta^{y_2''} = \alpha^{y_1'}\beta^{y_2'} \pmod p$$

$$\alpha^{y_1''+a_0y_2''} \equiv \alpha^{y_1'+a_0y_2'} \pmod p$$

因$p=2q+1$,有:

$$y_1''+a_0y_2'' \equiv y_1'+a_0y' \pmod q$$

$$y_1''-y_1' \equiv a_0(y_2'-y_2'') \pmod q$$

$$a_0 = \log_\alpha\beta = (y_1''-y_1')(y_2'-y_2'')^{-1} \pmod q$$

这样,签名者就能通过a_0出具一份伪造证明。

例 6.12 在Fail-stop签名方案中,已知$p=1019$,$q=509$,$p=2q+1$,取:

$$\alpha = 11, \quad \beta = 373$$

相应的私钥为:

$$a_0 = 23$$

选取:

$$a_1 = 408, \quad a_2 = 302$$

$$b_1 = 223, \quad b_2 = 199$$

可计算出:

$$\gamma_1 = 11^{408} \times 373^{302} \pmod{1019} = 124$$

$$\gamma_2 = 11^{223} \times 373^{199} \pmod{1019} = 961$$

消息为:

$$x = 200$$

现有一个签名,不知其真伪:

$$y'' = (262,398)$$

虽然也能够通过验证:

$$ver_K(x,y'') = \mathrm{true} \Leftrightarrow 124 \times 961^{200} = 11^{262} \times 373^{398} = 565 \pmod{1019}$$

但实际上它是伪造的,接下来签名方将证明它是伪造的。

签名方可通过自己的签名算法计算出x的真实签名:

$$y' = (408 + 200 \times 223, 302 + 200 \times 199) \bmod 509 = (216, 400)$$

对 y' 进行验证：

$$ver_K(x, y') = \text{true} \Leftrightarrow 124 \times 961^{200} = 11^{216} \times 373^{400} = 565 \pmod{1019}$$

接下来是对 y'' 进行伪造判断，由于：

$$a_0 = (262 - 216)(400 - 398)^{-1} \pmod{509} = 23$$

所以说明 y'' 是伪造的。

Fail-stop 签名方案还是一种一次签名方案，因为如果用同一个密钥 K 对两个消息进行签名，那么会产生以下方程组：

$$y_1 \equiv a_1 + x_1 b_1 \pmod q$$
$$y_2 \equiv a_2 + x_1 b_2 \pmod q$$
$$y'_1 \equiv a_1 + x_2 b_1 \pmod q$$
$$y'_2 \equiv a_2 + x_2 b_2 \pmod q$$

在已知 (x_1, y_1, y_2) 和 (x_2, y'_1, y'_2) 的前提下，可以解出 a_1、a_2、b_1、b_2。

一旦 Fail-stop 签名第一次被伪造后，系统所有的参加者都能够知道签名方案已被攻破，于是该签名宣布无效，这也正是 Fail-stop 名称的来历。

6.7　其他特殊签名方案简介

除前面介绍的数字签名外，还有一些在特殊场合下使用的签名方案，下面将对一些常见的特殊签名方案进行简单的介绍。

1. 群签名

群签名的思想最早由 Chaum 和 Heyst 于 1991 年提出，适用于以下的情形。

一个公司有多台连接到局域网上的计算机，公司的每个部门都有自己的打印机，打印机也连在局域网上。一般要求只有本部门的人员才被允许使用本部门的打印机，所以打印时必须使打印机确信用户在哪个部门工作。同时，公司希望不暴露使用者的身份，可是如果在当天结束时发现打印机使用得太频繁，管理者又能够指出是谁频繁使用了那台打印机，并做出处罚。

可以把上面的模型概括为群签名的 3 条基本性质：

(1) 只有群体内的成员才能够对消息进行签名；

(2) 接收者能够验证签名是该群体内的一个有效签名，但不能发现该签名具体是由哪一个成员所签；

(3) 如果出现争议，该签名可以被“打开”以查实签名者的身份。

从以上 3 条基本性质，可以得到群签名的若干技术需求。

(1) 防伪性：只有群成员才能产生有效的群签名。

(2) 匿名性：给定一个群签名后，除群管理员之外的任何人，确定签名人的身份在计算上是不可行的。

(3) 可跟踪性：群管理员在必要时可以打开一个签名以确定签名人的身份，而且签名

人不能阻止一个有效签名的打开。

（4）防陷害攻击：包括群管理员在内的任何人都不能以其他群成员的名义产生有效的群签名。

（5）不关联性：在不打开签名的情况下，确定两个不同的签名是否为同一个群成员所签在计算上是不可行的。

（6）抗联合攻击：即使一些群成员串通起来，也不能产生一个有效的不被跟踪的群签名。

一个群签名方案往往包括以下 5 个关键算法。

（1）创建（Setup）算法：通常是通过概率算法产生群公钥和群管理员的私钥。

（2）加入（Join）算法：一个用户和群管理员之间使用户成为群成员的协议。执行该协议可产生群成员的私钥和成员证书。

（3）签名（Sign）算法：输入是要签名的消息和一个群成员的私钥，输出是该消息的签名。

（4）验证（Verify）算法：输入是消息的签名和群公钥后，输出的结果是签名是否有效。

（5）打开（Open）算法：给定一个签名和群管理员的私钥，确定签名人的具体身份。

最经典的群签名方案莫过于 CS97 方案。1997 年 Camenisch 发表了论文 *Efficient and generalized group signature*，文中首次提出了知识签名的概念，并依据这种新思路给出了一个实用的群签名方案。由于引入了知识签名的方法，研究者们发现群签名验证的目的被明确，签名者必须在验证中应用知识签名清晰地证明自己拥有的有效身份，才能防止任何非授权者伪造签名。

由于篇幅限制，本书没有系统地介绍知识签名的相关原理，CS97 方案在此也一并略过，感兴趣的读者可查阅相关文献。

2. 盲签名

早在 1982 年 Chaum 就提出了盲签名的概念。

Chaum 给出一个例子：当文件装在信封中时，任何人都不能读它，签这个文件就是在信封里放一张复写纸，当签名者签这个信封时，他的签名便透过复写纸签到了文件上。

可以把盲签名的需求更明确地表达如下：

（1）消息的内容对签名者是盲的；

（2）即使签名者保存签过的文件，也不能确定出所签文件的真实内容。

针对以上需求，通常盲签名的实施包括以下几个步骤：

（1）在将原始信息发送给签名者前，首先要对原始信息进行盲化；

（2）签名者对盲化后的信息进行签名并返还给接收者；

（3）接收者得到签名后，先进行去盲化，然后得到签名者关于原始信息的正确签名。

盲数字签名的基本原理是实用两个公钥密码算法，第一个公钥密码算法用于隐蔽信息，可称为盲变换，第二个公钥密码算法用于签名。

下面以 RSA 盲签名为例进行说明。

（1）Alice 使用随机数 r 和 Bob 的公钥(n,b)，构造盲消息$\overline{m}=f(m)r^b \bmod n$。

（2）Bob 收到$\overline{m}$后便利用自己的私钥 a 产生签名$\overline{s}=\overline{m}^a \bmod n$。

(3) Alice 通过计算$\bar{s}r^{-1}=(mr^b)^a r^{-1}=m^a=s$ 获得 Bob 关于消息 m 的签名。

(4) Alice 通过验证 $s^a=f(m)\bmod n$ 来验证签名的有效性。

可以看出，由于盲因子 r 的作用，Bob 从$\bar{m}$中看不到真实消息 m，但 Alice 却可利用 r 获得签名。

盲签名是电子现金系统和电子投票系统中的关键技术。

3. 代理签名

在日常生活中，经常需要将自己的某些权力委托给可靠的代理人，由代理人代表本人去行使签名权，代理签名是这一需求的解决方案。

可以把代理签名的需求更明确地总结如下。

(1) 不可伪造性：除了原始签名者，只有指定的代理签名者能够代表原始签名者产生有效的签名。

(2) 可验证性：从代理签名中，验证者能够确信原始签名者认同了这份签名消息。

(3) 不可否认性：一旦代理签名者代替原始签名者产生了有效的代理签名，就不能向原始签名者否认他所签的有效代理签名。

(4) 可区分性：任何人都可区分代理签名和原始签名者的签名。

(5) 可识别性：原始签名者能够从代理签名中确定代理签名者的身份。

通常代理签名的实施包括以下几个步骤：

(1) 初始化过程。确定原始签名者、代理签名者的密钥。

(2) 权力委托过程。原始签名人利用他的私钥计算出一个数 σ，然后将 σ 秘密地交给代理签名者。任何人(包括代理签名者)都无法通过 σ 去破解原始签名人的私钥。

(3) 代理签名的生成过程。代理签名者可以利用 σ 和自己的私钥，生成一个新的签名密钥 θ，代理人用 θ 进行签名。

(4) 代理签名的验证过程。通过原始签名人的公钥，可以验证签名的有效性。

在代理签名的过程中，σ 被称为委托密钥，θ 则是代理签名密钥。

1996 年 Mambo、Usuda 和 Okamoto 提出的基于 ElGamal 签名体制的代理签名方案是一个典型的代理签名方案。

下面简要说明 MUO 方案的原理。

设 p 是一个大素数，q 是 $p-1$ 的一个素因子，$g\in Z_p^*$ 是一个 q 阶生成元。

原始签名方的私钥是 $x_0\in Z_q$，相应的公钥是：

$$y_0 = g^{x_0} \bmod p$$

原始签名方随机选择 $r\in Z_q$，计算：

$$R = g^r \bmod p,\quad \sigma = x_0 + rR \bmod q$$

原始签名方将(R,σ)发送给代理签名方。

代理签名方检验等式：

$$g^\sigma \equiv y_0 R^R \bmod p$$

若等式成立，则 σ 是有效的委托密钥；否则，拒绝接受原始签名方的代理授权。

代理签名方使用代理签名密钥 σ 对消息 M 进行签名。

验证者首先计算代理签名公钥：

$$y' \equiv y_0 R^R \bmod p$$

然后通过相应的验证算法进行验证。

代理签名是电子支付系统中的关键技术之一。

6.8 习 题

1. 在 RSA 签名方案中，取 $n=59\ 251$，$a=28\ 151$，试对消息 $x=2005$ 进行签名。

2. 在 RSA 签名方案中，取 $p=193$，$q=307$，$b=1223$，试验证以下签名是否合法：

(1) (192，1543)

(2) (961，361)

(3) (513，20 800)

3. 在 ElGamal 签名方案中，设 $p=1321$，$\alpha=13$，$a=211$，试对消息 $x=999$ 进行 ElGamal 签名。

4. 在 ElGamal 签名方案中，设 $p=1321$，$\alpha=13$，$\beta=449$，试验证下列签名(γ,δ,x)是否为合法签名：

(1) (604，810，486)

(2) (605，830，586)

(3) (604，850，686)

5. 在 ElGamal 签名方案中，设 $p=1321$，$\alpha=13$，$\beta=633$，在不知道$\log_\alpha\beta$ 的前提下试构造出一个合法签名。

6. 在 ElGamal 签名方案中，设 $p=1321$，$\alpha=13$，$\beta=1270$，并且已知一组签名：

$$(\gamma,\delta,x)=(666,515,131)$$

试构造出另一个合法签名。

7. 在 ElGamal 签名方案中，设 $p=1321$，$\alpha=13$，$\beta=799$，$\log_\alpha\beta$ 未知，并已知两个签名分别为：

$$(\gamma,\delta_1,x_1)=(604,762,666)$$
$$(\gamma,\delta_2,x_2)=(604,165,717)$$

而且两个签名使用了同一个 t，试求 t 值。

8. 对消息 $x=(1,0,0)$进行 Lamport 签名，具体参数为 $p=8447$，$\alpha=29$ 是生成元。

$$y=\begin{bmatrix}419 & 718\\731 & 825\\725 & 835\end{bmatrix}$$

9. 判断下列哪些签名是合法的 Lamport 签名，具体参数为 $p=9001$，$\alpha=23$ 是生成元，y 未知。

$$z=\begin{bmatrix}4555 & 8413\\955 & 3798\\8047 & 5021\end{bmatrix}$$

(1) $x=(1,0,1)$，$sig(x)=(718,731,835)$

(2) $x=(1,1,0)$, $sig(x)=(718,825,725)$

(3) $x=(0,0,1)$, $sig(x)=(419,825,725)$

10. 对消息 $x=1200$ 实施 Chaum-van Antwerpen 签名，具体参数为 $p=3023$，$q=1511$，$\alpha=23$，$a=201$。

11. 在 Chaum-van Antwerpen 签名方案中，具体参数为 $p=3023$，$q=1511$，$\alpha=23$，$\beta=2102$，试分别详细说明下列签名的验证过程：

(1) $x=100$，$sig(x)=2870$，$e_1=93$，$e_2=67$

(2) $x=250$，$sig(x)=2900$，$e_1=93$，$e_2=67$

(3) $x=250$，$sig(x)=2901$，$e_1=193$，$e_2=167$

12. 在 Chaum-van Antwerpen 签名方案中，具体参数为 $p=3023$，$q=1511$，$\alpha=23$，$\beta=2102$，$e_1=93$，$e_2=67$，试根据返回的 d 的结果判断下列签名是否为有效签名：

(1) $x=846$，$sig(x)=1592$，$d=1378$

(2) $x=825$，$sig(x)=1240$，$d=2772$

(3) $x=608$，$sig(x)=42$，$d=3496$

13. 对消息 $x=200$ 实施 Fail-stop 签名，具体参数为 $p=12\ 107$，$q=6053$，$\alpha=29$，$a=509$，$a_1=718$，$a_2=731$，$b_1=419$，$b_2=725$。

14. 在 Fail-stop 签名方案中，具体参数为 $p=10\ 079$，$q=5039$，$\alpha=23$，$\beta=2542$，$a_1=408$，$a_2=302$，$b_1=223$，$b_2=199$，试验证下面签名是否为有效签名：

(1) $x=100$，$sig(x)=(2552,46)$

(2) $x=250$，$sig(x)=(729,4700)$

(3) $x=1000$，$sig(x)=(1692,2781)$

第 7 章　Hash 函数与消息认证码

一般情况下，数字签名的长度往往比消息长，而现实中的长消息又非常普遍，因此数字签名算法实现过程中的操作难度将非常大。不可能对整条长消息直接签名，因为超过有限域的阶的数据就意味着无法处理。如果将长消息分成若干段后分别签名，那样做非常麻烦，而且在拼接的时候又会产生数据完整性问题。比较合理的做法是在数字签名前对消息先进行数字摘要，然后再对数字摘要进行签名。

数字摘要将任意长度的消息变成固定长度的短消息，类似于一个自变量是消息的函数，这个函数是 Hash 函数。

7.1　Hash 函数的基本概念

首先给出一个 Hash 函数的定义。

Hash 函数(族)：一个 Hash 函数(族)是一个三元组(X,Y,H)，其中：

(1) X 表示可能的消息集合；

(2) Y 表示一个有限集合；

(3) 存在 $h\in H$，使得 $h: X\to Y$。

X 是有限集合还是无限集合并没有要求。当 X 是有限集合时，Hash 函数就成了压缩算法。

一般情况下，约定$|X|\geqslant|Y|$，大多数情况下甚至约定$|X|\geqslant 2|Y|$。

给定 X 和 Y，令 $F(X,Y)$表示由 X 映射到 Y 的所有可能的函数，称 $F(X,Y)$是 Hash 函数。

不妨设$|X|=N$，$|Y|=M$，那么$|F(X,Y)|=M^N$，有时也称 $F(X,Y)$是一个规模是(N,M)的 Hash 函数族。

Hash 函数与 Hash 函数族的概念有时候并不明确，读者可根据上下文来区分。

接下来的 3 个概念对于 Hash 函数的安全性分析至关重要。

(1) **原像问题**：给定一个 Hash 函数 h，y 为一个消息摘要，寻找出 x 使得 $y=h(x)$的问题称为原像问题。

给定一个 Hash 函数 h，y 为一个消息摘要，若要找出 x 使得 $y=h(x)$在计算上不可行，则称此 Hash 函数为抗原像攻击的。

(2) **第二原像问题**：给定 Hash 函数 h 和任意给定的消息 m，寻找一个 m'，$m'\neq m$，使得 $h(m')=h(m)$ 的问题称为第二原像问题。第二原像问题也称次原像问题。

给定 Hash 函数 h 和任意给定的消息 m，如果要找一个 m'，$m'\neq m$，使得 $h(m')=h(m)$在计算上不可行，则称 h 是抗第二原像攻击的，也称该 Hash 函数是弱无碰撞(Weak Collision-Free)的。

(3) 碰撞问题：寻找两个消息 m_1、m_2，$m_1 \neq m_2$，使得 $h(m_1)=h(m_2)$，这个问题称为碰撞(Collision)问题。

如果有两个消息 m_1、m_2，$m_1 \neq m_2$，使得 $h(m_1)=h(m_2)$，就说这两个消息 m_1 和 m_2 是碰撞的消息。

如果要找到任意一对消息 m_1、m_2，$m_1 \neq m_2$，使得 $h(m_1)=h(m_2)$在计算上不可行，则称 h 是抗碰撞攻击的。该 Hash 函数又称为强无碰撞(Strong Collision-Free)的 Hash 函数。

为了分析 Hash 函数的安全性，先介绍一种理想的 Hash 函数模型。

这个理想的 Hash 函数模型又被称为**随机预言模型**(Random Oracle Model)。在随机预言模型中，无法给出一个公式或算法来计算函数 h 的值。要得到计算 $h(x)$的唯一方法是询问随机预言器。询问随机预言器的做法类似于查询一个很大的表。

对于每个 x，有一个完全随机的值 $h(x)$与之对应。

这种随机性能够带来很大的安全性，即使已经知道很多$(x_1,h(x_1))$，$(x_2,h(x_2))$，…，$(x_k,h(x_k))$，也无法计算出下一个 x 对应的$h(x)$，要得到 $h(x)$只能是询问随机预言器。

为了理解随机的重要性，下面举一个相反的例子。

假设构造这样一个 Hash 函数：$Z_n \times Z_n \rightarrow Z_n$

$$h(x,y) = ax + by \bmod n$$

其中，a、$b \in Z$，是一个大于 2 的素数。

现在得到两组 Hash 函数值，分别是$((x_1,y_1),h(x_1,y_1))$和$((x_2,y_2),h(x_2,y_2))$。

不妨设 $z_1=h(x_1,y_1)$，$z_2=h(x_2,y_2)$。

假设$(x_3,y_3)=(px_1+qx_2,py_2+qy_2)$，那么可以计算出：

$$h(x_3,y_3) = a(px_1 + qx_2) + b(py_2 + qy_2) = pz_1 + qz_2$$

于是，可以通过以往的 Hash 函数计算出新的 Hash 函数。

随机预言模型则不是这样，一个新的 x 的 Hash 函数值只能通过询问随机预言器获得，无法从已经得到的 Hash 函数对中获得任何帮助。

定理 7.1 假定 $h \in F(X,Y)$是随机选择的，令 $X_0 \subseteq X$、$x \in X_0$ 时的 $h(x)$值通过查询 h 的预言器已经确定，那么对所有的 $x \in X \backslash X_0$ 和所有的 $y \in Y$，则有 $P(h(x)=y)=1/M$，其中 $M=|Y|$。

上面的概率实际上是已知一些 Hash 值的条件概率。

对于随机预言模型，求解原像问题只能用下面的算法：

已知 $X_0 \subseteq X$，且$|X_0|=Q$。

```
for each x∈ X0 do
  if (h(x) = y) then return x;
return failure.
```

这个算法是一种(ε,Q)拉斯维加斯(Las Vegas)算法。

用拉斯维加斯算法找到一个解，这个解就一定是正确解。但用拉斯维加斯算法不一定能找到解。对于所求解问题，用同一拉斯维加斯算法反复对该实例求解足够多次，可使求解失败的概率小于 ε。

这里求解原像问题的成功概率可简单求出：

对于 $1 \leqslant i \leqslant Q$，设 E_i 表示第 i 次查询成功的事件，即 $h(x_i)=y$ 成立的事件。根据前面

的定理,这些独立的事件概率 $P(E_i)=1/M$,于是有:

$$P(E_1 \cup E_2 \cup \cdots \cup E_Q)=1-(1-1/M)^Q$$

当 Q 远小于 M 时,$P(E_1 \cup E_2 \cup \cdots \cup E_Q) \approx Q/M$。

类似地,也可以求解第二原像问题,算法如下:

已知 $y=h(x)$,$X_0 \subseteq X \backslash \{x\}$,显然 $|X_0|=Q-1$。

```
for each x0 ∈ X0 do
    if(h(x0) = y) then return x0;
return failure.
```

同理,求解第二原像问题成功的概率是:

$$P(E_1 \cup E_2 \cup \cdots \cup E_{Q-1}) = 1-(1-1/M)^{Q-1}$$

求解碰撞问题的求解方法,也可以类似地给出:

已知 $X_0 \subseteq X$,且 $|X_0|=Q$。

```
for each x∈ X0 do
  y = h(x)
if (y = h(x')) ( for some x')
   then return(x, x’)
else return failure.
```

碰撞问题求解成功的概率要复杂一些,为此这里引入一个概率问题——生日悖论。

生日悖论来自于生日问题,生日问题是指在多少学生中,能够保证至少两个学生的生日在同一天的概率不小于 1/2?

答案是 23,这就是所谓的生日悖论。这里的悖论只是沿用以往的说法而已,并不是真正的悖论,只不过因为正确答案不是很直观。

生日问题和 Hash 函数的碰撞问题有类似之处。

设 $h: X \to Y$ 是一个 Hash 函数,X 和 Y 都是有限的,并且 $|X| \geqslant 2|Y|$,记 $|X|=m$,$|Y|=n$。

显然碰撞是不可避免的,至少会有 n 个碰撞,问题在于如何确定碰撞发生的概率。一个直观的办法是随机选择 k 个不同的元素 x_1、x_2、…、$x_k \in X$,计算 $y_i=h(x_i)(1 \leqslant i \leqslant k)$,然后确定有多少个碰撞事件发生。

这个过程类似于将 k 个球随机扔进 n 个筐子中,然后检查是否出现某个筐子有两个球的事件。

下面计算该事件发生概率的下界。

将 k 个球随机依次扔进 n 个筐子后,不发生任何碰撞的概率是:

$$\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\cdots\left(1-\frac{k-1}{n}\right)=\prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right)$$

将 e^{-x} 展开成级数形式:

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!}\cdots$$

当 x 很小时,可以做近似处理:

$$1 - x \approx e^{-x}$$

于是:

$$\prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right)\approx\prod_{i=1}^{k-1}\mathrm{e}^{-\frac{i}{n}}=\mathrm{e}^{-\frac{k(k-1)}{n}}$$

因此，至少有一次碰撞发生的概率约为：

$$1-\mathrm{e}^{-\frac{k(k-1)}{n}}$$

设至少发生一次碰撞的概率为 ε，则：

$$\mathrm{e}^{-\frac{k(k-1)}{n}}\approx 1-\varepsilon$$

解出 k 得：

$$k\approx\sqrt{n\ln\frac{1}{1-\varepsilon}}$$

令 $\varepsilon=0.5$ 可得：

$$k\approx 1.17\sqrt{n}$$

再令 $n=365$，可得：

$$k\approx 22.3$$

于是证明了前面的生日悖论。

生日悖论给出了安全 Hash 函数的一般要求，对于数字摘要来说，通常 40 位长的数字摘要被认为是不安全的。因为随机计算 $1.17\times2^{20}\approx120$ 万次的数字摘要，将以 1/2 的概率产生碰撞。

7.2　Hash 函数的构造

前面介绍了 Hash 函数的安全性，在构造 Hash 函数时，往往希望构造出的 Hash 函数能够是强无碰撞的。

虽然无法构造出真正的符合随机预言模型的 Hash 函数，但构造出强无碰撞的 Hash 函数是可能的。下面就给出一个基于离散对数问题构造出的 Hash 函数，这个 Hash 函数速度不快，以至于难以实际应用，但是是一个典型的强无碰撞的 Hash 函数。

Chaum-van Heijst-Pfitzmann Hash 函数：假设 p、q 为满足 $p=2q+1$ 的两个素数，令 α、β 为 Z_p 的两个本原元，$\log_\alpha\beta$ 不公开，构造 Hash 函数如下：

$$h(x_1,x_2)=\alpha^{x_1}\beta^{x_2}\bmod p$$

Chaum-van Heijst-Pfitzmann Hash 函数将 $\{0,1,\cdots,q-1\}\times\{0,1,\cdots,q-1\}$ 上的消息映射到 $\{1,2,\cdots,p-1\}$ 的 Hash 值上。

例 7.1　在 Chaum-van Heijst-Pfitzmann Hash 函数中，已知 $p=1019$，$q=509$，$p=2q+1$，取：

$$\alpha=16,\quad\beta=435$$

消息：

$$x=(23,37)$$

相应的摘要是：

$$h(x_1,x_2)=16^{23}\,435^{37}\bmod 1019=416$$

下面的定理说明 Chaum-van Heijst-Pfitzmann Hash 函数是强无碰撞函数。

定理 7.2 如果得到 Chaum-van Heijst-Pfitzmann Hash 函数的一个碰撞，那么 $\log_\alpha\beta$ 可根据该碰撞计算出来。

证明：假设 (x_1,x_2) 和 (x_3,x_4) 是 Chaum-van Heijst-Pfitzmann Hash 函数的一个碰撞，即：

$$(x_1,x_2)\neq(x_3,x_4)$$
$$h(x_1,x_2)=h(x_3,x_4)$$

那么：

$$\alpha^{x_1}\beta^{x_2}=\alpha^{x_3}\beta^{x_4}\bmod p$$
$$\alpha^{x_1-x_3}=\beta^{x_4-x_2}\bmod p$$

设 $d=\gcd((x_4-x_2),p-1)$，因为 $p=2q+1$，所以：

$$d\in\{1,2,q,2q\}$$

接下来根据 d 的 4 种取值分别讨论。

1) $d=1$

此时，(x_3,x_4) 与 $p-1$ 互素，因此 x_4-x_2 模 $p-1$ 的逆存在，不妨设：

$$y=(x_4-x_2)^{-1}\bmod(p-1)$$

则：

$$\beta=\beta^{(x_4-x_2)y}=\alpha^{(x_1-x_3)y}(\bmod\ p)$$

于是：

$$\log_\alpha\beta=(x_1-x_3)(x_4-x_2)^{-1}\bmod(p-1)$$

2) $d=2$

此时，因为 $p-1=2q$，(x_3,x_4) 与 q 互素，(x_4-x_2) 模 q 的逆存在，不妨设：

$$y=(x_4-x_2)^{-1}\bmod q$$

则：

$$(x_4-x_2)y=kq+1$$

又因为：

$$\beta^q=-1(\bmod\ p)$$

所以：

$$\beta^{(x_4-x_2)y}=\beta^{kq+1}=(-1)^k\beta=\pm\beta(\bmod\ p)$$

于是：

$$\alpha^{(x_4-x_2)y}=\beta^{(x_1-x_3)y}=\pm\beta(\bmod\ p)$$
$$\log_\alpha\beta=(x_1-x_3)y\bmod(p-1)$$

或

$$\log_\alpha\beta=(x_1-x_3)y+q\bmod(p-1)$$

3) $d=q$

因为 $0\leqslant x_1\leqslant q-1$，$0\leqslant x_3\leqslant q-1$，所以：

$$-(q-1)\leqslant x_4-x_2\leqslant(q-1)$$

这与 $\gcd((x_4-x_2),p-1)=q$ 产生矛盾，也就是说这种情况不可能出现。

4) $d=2q$

此时：

$$(p-1) \mid (x_4 - x_2)$$

于是：

$$x_4 = x_2$$
$$\alpha^{x_1} = \alpha^{x_3} \pmod p$$
$$x_1 = x_3$$

这与$(x_1,x_2)\neq(x_3,x_4)$矛盾，说明这种情况不可能出现。

总之，如果获得 Chaum-van Heijst-Pfitzmann Hash 函数的一个碰撞，就能够计算出 $\log_\alpha\beta$，也就是说寻找 Chaum-van Heijst-Pfitzmann Hash 函数的一个碰撞和离散对数问题的求解等价，基于这个事实，Chaum-van Heijst-Pfitzmann Hash 函数是强无碰撞 Hash 函数。

例 7.2　在 Chaum-van Heijst-Pfitzmann Hash 函数中，已知 $p=1019, q=509, p=2q+1$，取：

$$\alpha = 7, \quad \beta = 547$$

消息 1：

$$x = (421, 35)$$

相应的摘要是：

$$h(x_1, x_2) = 7^{421} \times 547^{35} \bmod 1019 = 267$$

消息 2：

$$x' = (23, 37)$$

相应的摘要是：

$$h(x'_1, x'_2) = 7^{23} \times 547^{37} \bmod 1019 = 267$$

于是(23,37)和(421,35)发生了碰撞。

$$d = \gcd(37-35, 1018) = 2$$

适用第(2)种情况：

$$y = (37-35)^{-1} \bmod 509 = 255$$
$$\log_7 547 = (421-23) \times 255 \bmod 1018 = 708$$

或

$$\log_7 547 = (421-23) \times 255 + 509 \bmod 1018 = 199$$

容易验证：$\log_7 547=199$ 是正解。

Chaum-van Heijst-Pfitzmann Hash 函数是一个不错的强无碰撞函数，但它的定义域是有限的，因此并不是完全符合数字摘要的要求。

在对消息做摘要时要求不受消息长度的约束，于是一种直观的想法是将已有的定义在有限定义域上的 Hash 函数“延伸”到无限的定义域上，即“扩展 Hash 函数”。

扩展 Hash 函数：设 $h:(Z_2)^m \to (Z_2)^t$ 是一个强无碰撞的 Hash 函数，可通过 h 构造扩展 Hash 函数：

$$h^*: X \to (Z_2)^t$$

先考虑 $m \geqslant t+2$ 的情况，设：

$$x = x_1 \parallel x_2 \parallel \cdots \parallel x_u \in X$$

其中，$|x_1|=|x_2|=|x_{u-1}|=m-t-1$。

$|x|$表示二进制串的长度。不难得出：

$$u = \left[\frac{n}{m-t-1}\right]$$

$$|x_k| = m-t-1-d, 0 \leqslant d \leqslant m-t-2$$

具体扩展方法如下：

```
(1) for i = 1 to u-1 do
(2) y_i = x_i
(3) y_u = x_u ‖ 0…0(通过补 0 使得|y_u| = m-t-1)
(4) 令 y_{u+1}是 d 的二进制串表示
(5) g_1 = h(0^{t+1} ‖ y_1)
(6) for i = 1 to u do
(7) g_{i+1} = h(g_i ‖ 1 ‖ y_{i+1})
(8) h*(x) = g_{u+1}
```

其中，$y(x)=y_1 \| y_2 \| \cdots \| y_{u+1}$。

第(2)步结束时，$y_u=x_u \| 0^d$；第(3)步结束时，$y_{u+1}=0\cdots \| d$ 的二进制表示。

以下的定理说明如果 h 是强无碰撞的，那么 h^* 也是强无碰撞的。

定理 7.3 设 $h:(Z_2)^m \to (Z_2)^t$ 是一个强无碰撞的 Hash 函数，$m \geqslant t+2$，那么由上面扩展方法得到的函数 $h^*:X \to (Z_2)^t$ 也是一个强无碰撞的 Hash 函数。

证明：采用反证法，证明的思路是如果函数 h^* 不是一个强无碰撞的 Hash 函数，那么可以找到：

$$x \neq x'$$

使得：

$$h^*(x) = h^*(x')$$

在这种情况下，如果能够用这个 x 或 x'构造出 h 的一个碰撞，就产生了矛盾，因为 h 是强无碰撞的 Hash 函数。

将 $y(x)$和 $y(x')$写成串的形式：

$$y(x) = y_1 \| y_2 \| \cdots \| y_{u+1}$$

$$y(x') = y'_1 \| y'_2 \| \cdots \| y'_{v+1}$$

其中，x 和 x'分别在第(4)步级联了 d 和 d'个 0。

设在第(5)步和第(7)步的计算结果分别为 g_1、g_2、…、g_{u+1}和 g'_1、g'_2、…、g'_{v+1}。

如果$|x| \neq |x'| \bmod (m-t-1)$，则 $d \neq d'$，于是：

$$y_{u+1} \neq y'_{v+1}$$

而：

$$h(g_u \| 1 \| y_{u+1}) = g_{u+1} = h^*(x) = h^*(x') = g'_{v+1} = h(g'_v \| 1 \| y'_{v+1})$$

所以 $g_u \| 1 \| y_{v+1}$ 与 $g'_v \| 1 \| y'_{v+1}$是一对碰撞。

如果$|x| = |x'| \bmod (m-t-1)$，分以下两种情况讨论。

1) $|x| = |x'|$

此时，$u=v$ 且 $y_{u+1} \neq y'_{u+1}$。因为：

$$h(g_u | 1 \| y_{u+1}) = g_{u+1} = h^*(x) = h^*(x') = g'_{v+1} = h(g'_v \| 1 \| y'_{v+1})$$

如果 $g_u \neq g'_u$，那么 $g_u \| 1 \| y_{v+1}$ 与 $g'_v \| 1 \| y'_{v+1}$是一对碰撞。

如果 $g_u = g'_u$，那么：

$$h(g_{u-1} \parallel 1 \parallel y_u) = g_u = g'_u = h(g'_u \parallel 1 \parallel y'_u)$$

如果 $g_{u-1} \neq g'_{u-1}$ 或 $y_u \neq y'_u$，h 的碰撞就产生了，否则继续往下推，直到最后：

$$h(0^{t+1} \parallel y_1) = g_1 = g'_1 = h(0^{t+1} \parallel y'_1)$$

如果 $y_1 \neq y'_1$，h 的碰撞产生。

不可能对所有 $1 \leqslant k \leqslant u+1$，都有 $y_k \neq y'_k$，因为那样的话

$$y(x) = y(x')$$

也说明 $x = x'$，与反证法的假设不符。

于是，这种情况下总能找到 h 的一个碰撞。

2) $|x| \neq |x'|$

不失一般性，假设 $|x'| > |x|$，则 $v > u$。

类似于(1)去寻找碰撞，如果按照该方法没有找到碰撞，则必有：

$$h(0^{t+1} \parallel y_1) = g_1 = g'_{v-u+1} = h(g'_{v-u} \parallel 1 \parallel y'_{v-u+1})$$

但 $0^{t+1} \parallel y_1$ 的第 $t+1$ 的比特是 0，而 $g'_{v-u} \parallel 1 \parallel y'_{v-u+1}$ 的第 $t+1$ 比特是 1，所以 $0^{t+1} \parallel y_1$ 与 $g'_{v-u} \parallel 1 \parallel y'_{v-u+1}$ 是一对碰撞。

命题得证。

前面的方法、定理都是针对 $m \geqslant t+2$ 的情况。当 $m = t+1$ 时，扩展 Hash 函数的构造方法与 $m \geqslant t+2$ 时的构造方法有所不同。

设 $|x| = n > m$，首先对 x 进行一种特殊的编码，编码由相应的函数实现。

定义函数：

$$f(0) = 0$$
$$f(1) = 01$$

具体扩展方法如下：

```
(1) 令 y = y1y2…yu = 11 ‖ f(x1) ‖ f(x2) ‖ … ‖ f(xn)
(2) g1 = h(0^t ‖ y1)
(3) for i = 1 to u-1 do
(4) g(i+1) = h(gi ‖ y(i+1))
(5) h*(x) = gu
```

在第(1)步进行的 $x \rightarrow y$ 编码中，$y(x)$ 满足两个重要性质：

(1) 如果 $x \neq x'$，那么 $y(x) \neq y(x')$；

(2) 不存在两个串 x 和 x'($x \neq x'$)和一个串 Z 使得：

$$y(x) = Z \parallel y(x')$$

即没有编码是另一个编码的后缀，原因是每个 $y(x)$ 串都是从 11 开始，之后串中不会再出现 11。

以下的定理说明如果 h 是强无碰撞的，那么 h^* 也是强无碰撞的。

定理 7.4　设 $h:(Z_2)^{t+1} \rightarrow (Z_2)^t$ 是一个强无碰撞的 Hash 函数，那么由上面扩展方法得到的函数 $h^*: X \rightarrow (Z_2)^t$ 也是一个强无碰撞的 Hash 函数。

证明：还是采用反证法。

证明的思路是如果函数 h^* 不是一个强无碰撞的 Hash 函数，那么可以找到 $x \neq x'$，使得 $h^*(x) = h^*(x')$。在这种情况下，如果能够用这个 x 或 x' 构造出 h 的一个碰撞，就产生了

矛盾,因为 h 是强无碰撞的 Hash 函数。

将 $y(x)$ 和 $y(x')$ 写成串的形式:

$$y(x) = y_1 \parallel y_2 \parallel \cdots \parallel y_{u+1}$$
$$y(x') = y'_1 \parallel y'_2 \parallel \cdots \parallel y'_{v+1}$$

分以下两种情况来分析。

(1) $u=v$。

类似于定理 7.3 同阶段的证明,要么能找到 h 的一个碰撞,要么最终得出 $x=x'$,与反证法的假设产生矛盾。

(2) $u \neq v$。

不失一般性,假设 $v>u$,类似与(1)的处理方法,可能找到 h 的一个碰撞。

如果没有找到碰撞,必有下列等式成立:

$$y_u = y'_v$$
$$y_{u-1} = y'_{v-1}$$
$$\cdots$$
$$y_1 = y'_{v-u+1}$$

这与编码规则的第(2)个重要性质矛盾,所以一定能找到 h 的一个碰撞。

命题得证。

可以将定理 7.3 和定理 7.4 合并成一个完善的定理。

定理 7.5 设 $h:(Z_2)^m \to (Z_2)^t$ 是一个强无碰撞的 Hash 函数,$m \geq t+1$,那么由扩展方法得到的函数 $h^*:X \to (Z_2)^t$ 也是一个强无碰撞的 Hash 函数。

7.3 具有迭代结构的 Hash 算法

7.2 节介绍了把一个有限定义域的 Hash 函数扩展到无限定义域的方法。不过,从具体算法来看,Chaum-van Heijst-Pfitzmann Hash 函数并不高效。

在实际的 Hash 函数构造中,具有迭代结构的构造方法占主流,虽然还有众多可选的结构,如基于校验和(Checksum)的结构、基于扩展图(Expanded Graphs)的结构等。

现有的具有迭代结构的哈希函数大都是基于 MD4 的设计准则衍生而来的,因此通常称这些函数为 MD4 系列。MD4 系列又可以进一步细分为 3 个子系列:MD 系列,SHA 系列和 RIPEMD 系列。

MD 系列主要包括 MD4、MD5 和 HAVAL 等算法。

SHA 系列主要包括 SHA-0、SHA-1 和 SHA-2(SHA-224/256 /384/512)等算法。

RIPEMD 系列主要包括 RIPEMD、RIPEMD-128 和 RIPEMD-160 等算法。

下面介绍 MD 系列中的 MD5 算法。

MD5 算法是由 Rivest(RSA 中的 R)于 1991 年提出的 Hash 算法,是目前最常用的 Hash 算法之一。MD5 算法是 MD4 算法的改进。

在 MD5 算法中,将消息划分成 512 位的消息块进行处理,最终形成 128 位的信息摘要。MD5 算法的总体流程如图 7.1 所示。

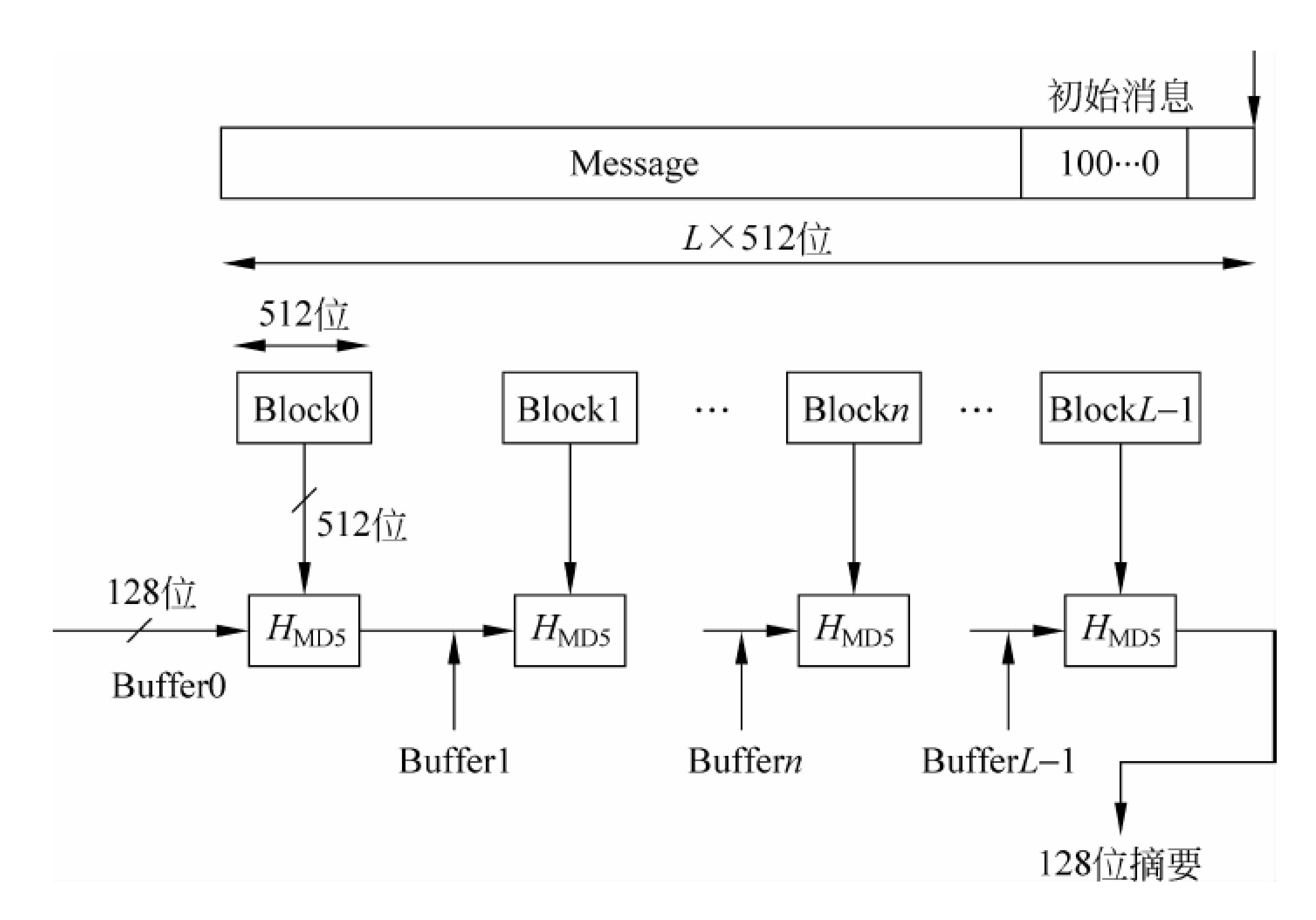


图 7.1　MD5 算法流程

下面根据 RFC1321，分 5 个步骤实现 MD5。

第 1 步，补位。补位的目标是使输入的消息长度从任意值变成一个新的长度 n，使得：

$$n = 448 (\text{mod } 512)$$

即通过补位使新消息的长度差 64 位成为 512 的整倍数。

即使原消息的长度正好满足要求，也需要进行补位。

补位的补丁在原消息之后包括一个 1，剩下全是 0。如果原消息的长度正好满足要求，则补位包括一个 1 和 511 个 0。

第 2 步，追加长度。在追加长度前，通过补位，消息长度已经变成模 512 余 448，接下来的追加长度将在消息后继续补充 64 位的信息，新消息将是 512 的整数倍，可按 512 位进行分组。

追加的长度信息由 64 位表示，被追加到已补的信息后，如果原消息长度超过 64 位，只使用低 64 位即可。

追加的长度是原消息的长度，而不是补位后的信息长度。

第 3 步，缓冲区初始化。为了计算 Hash 函数的结果，首先需要设置 128 位缓冲区。缓冲区除接受 Hash 函数最终结果外，还记录中间结果。

在图 7.2 中，将缓冲区分成 4 等份，即 4 个 32 位寄存器(A、B、C、D)，每个 32 位寄存器也被称为字(word)。

赋初值：

word A：01 23 45 67
word B：89 ab cd ef
word C：fe dc ba 98
word D：76 54 32 10

ABCD 构成 Buffer 0。

第 4 步，缓冲区 $n \to n+1$。从 Buffer 0 开始，将进行 4 回合运算实现缓冲区从 n 到 $n+1$ 的迭代。

4 回合运算是四组运算，而非四次运算，具体的运算过程如图 7.3 所示。图中的 CLS 表示移位。

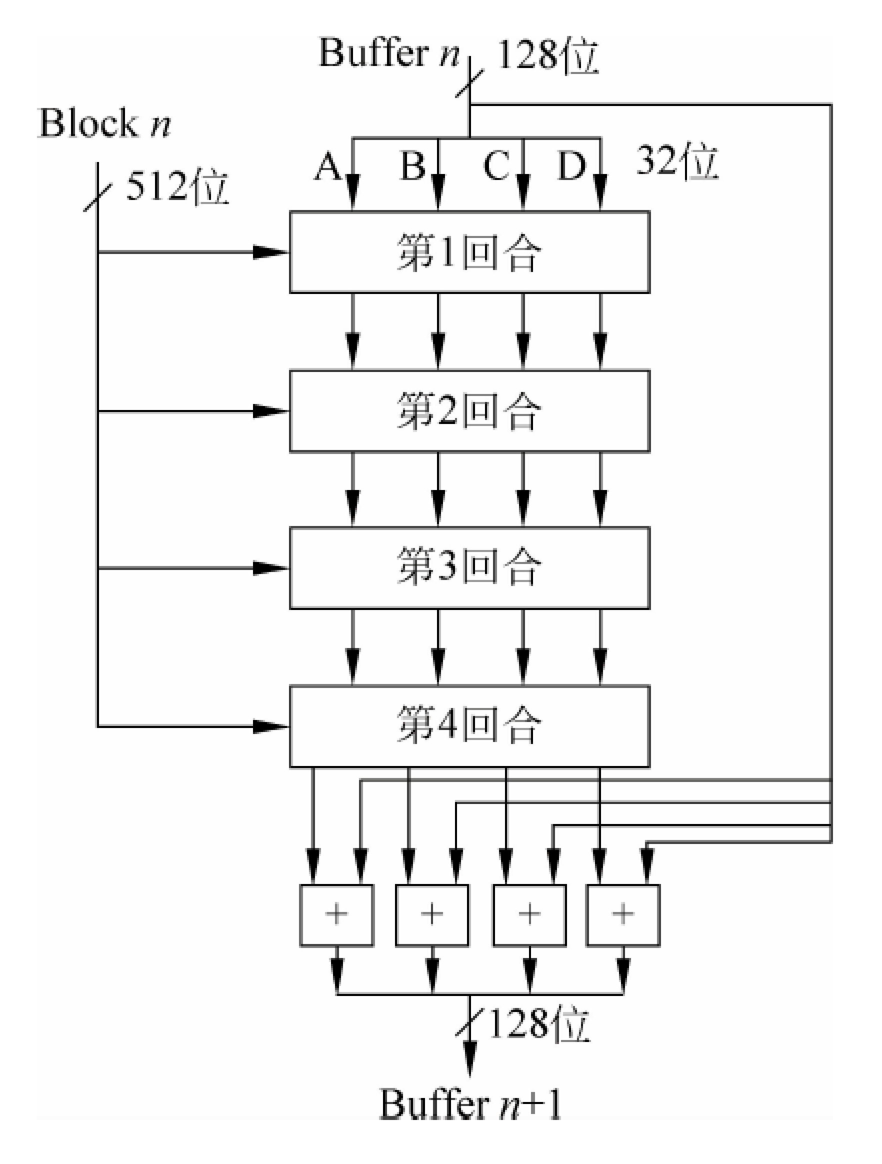


图 7.2 缓冲区 $n \to n+1$ 示意图

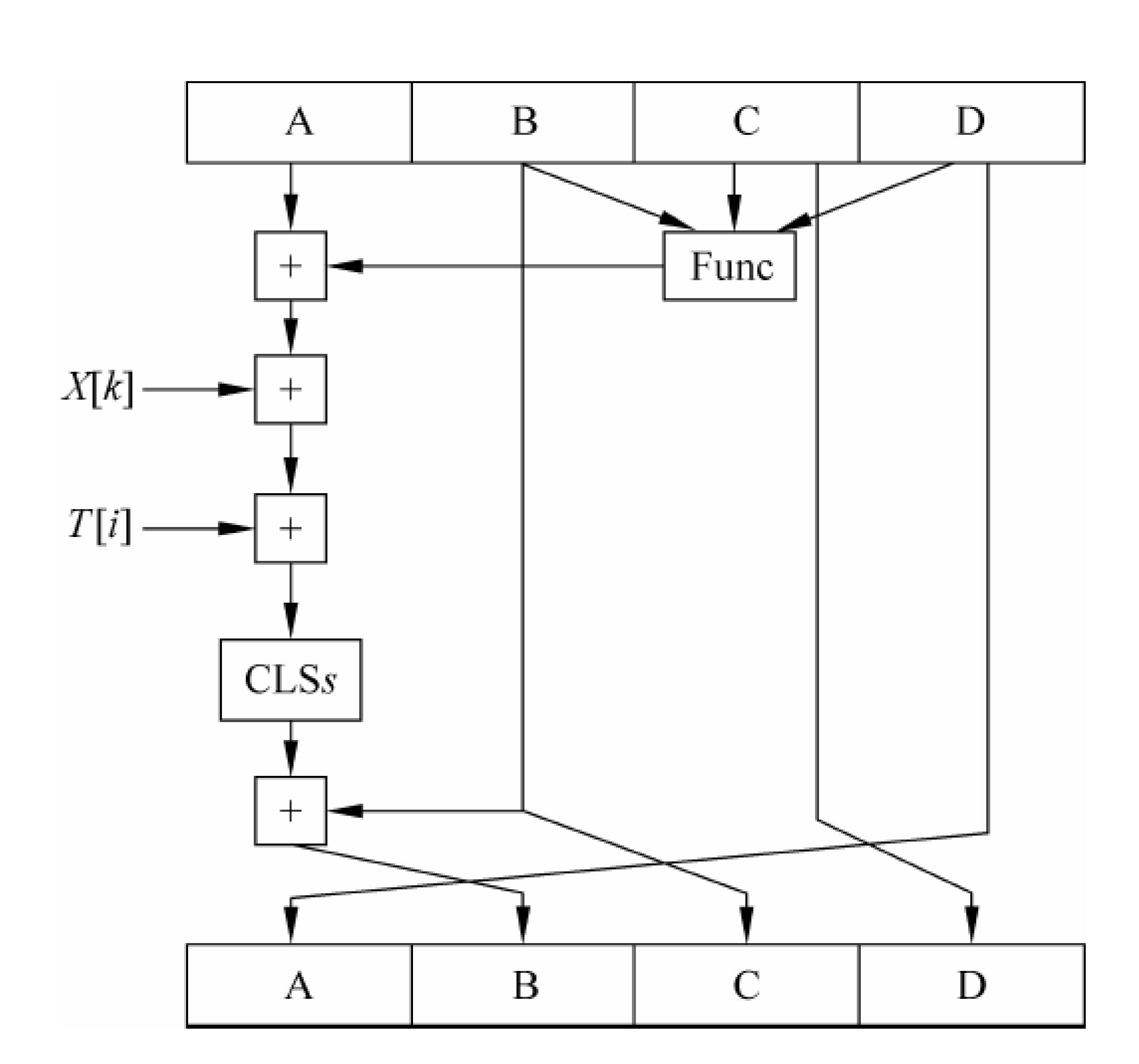


图 7.3 回合运算

在 4 回合运算中，函数 Func(a,b,c,d)各不相同，分别为 $F(X,Y,Z)$、$G(X,Y,Z)$、$H(X,Y,Z)$和 $I(X,Y,Z)$：

$$F(X,Y,Z) = (X \text{ and } Y) \text{ or } (\text{not}(X) \text{ and } Z)$$

$$G(X,Y,Z) = (X \text{ and } Z) \text{ or } (Y \text{ and } \text{not}(Z))$$

$$H(X,Y,Z) = X \text{ xor } Y \text{ xor } Z$$

$$I(X,Y,Z) = Y \text{ xor } (X \text{ or } \text{not}(Z))$$

图 7.2 中缓冲区从 n 到 $n+1$ 的迭代算法描述如下：

```
For i = 0 to N/16 - 1 do
    For j = 0 to 15 do
    设置 X[j]为 M[i * 16 + j]
    end
AA = A
BB = B
CC = C
DD = D
(第 1 回合)
设[abcd,k,s,i]表示寄存器运算: a = b +  ((a + F(b,c,d) + X[k] + T[i])<<< s)
[ABCD,0,7,1]        [DABC,1,12,2]        [CDAB,2,17,3]        [BCDA,3,22,4]
[ABCD,4,7,5]        [DABC,5,12,6]        [CDAB,6,17,7]        [BCDA,7,22,8]
[ABCD,8,7,9]        [DABC,9,12,10]       [CDAB,10,17,11]      [BCDA,11,22,12]
[ABCD,12,7,13]      [DABC,13,12,14]      [CDAB,14,17,15]      [BCDA,15,22,16]
(第 2 回合)
设[abcd,k,s,i]表示寄存器运算: a = b + ((a + G(b,c,d) + X[k] + T[i])<<< s)
[ABCD,1,5,17]       [DABC,6,9,18]        [CDAB,11,14,19]      [BCDA,0,20,20]
[ABCD,5,5,21]       [DABC,10,9,22]       [CDAB,15,14,23]      [BCDA,4,20,24]
```

```
[ABCD,9,5,25]     [DABC,14,9,26]    [CDAB,3,14,27]    [BCDA,8,20,28]
[ABCD,13,5,29]    [DABC,2,9,30]     [CDAB,7,14,31]    [BCDA,12,20,32]
(第 3 回合)
设[abcd,k,s,t]表示寄存器运算: a = b + ((a + H(b,c,d) + X[k] + T[i])<<< s)
[ABCD,5,4,33]     [DABC,8,11,34]    [CDAB,11,16,35]   [BCDA,14,23,36]
[ABCD,1,4,37]     [DABC,4,11,38]    [CDAB,7,16,39]    [BCDA,10,23,40]
[ABCD,13,4,41]    [DABC,0,11,42]    [CDAB,3,16,43]    [BCDA,6,23,44]
[ABCD,9,4,45]     [DABC,12,11,46]   [CDAB,15,16,47]   [BCDA,2,23,48]
(第 4 回合)
设[abcd,k,s,t]表示寄存器运算: a = b + ((a + I(b,c,d) + X[k] + T[i])<<< s)
[ABCD,0,6,49]     [DABC,7,10,50]    [CDAB,14,15,51]   [BCDA,5,21,52]
[ABCD,12,6,53]    [DABC,3,10,54]    [CDAB,10,15,55]   [BCDA,1,21,56]
[ABCD,8,6,57]     [DABC,15,10,58]   [CDAB,6,15,59]    [BCDA,13,21,60]
[ABCD,4,6,61]     [DABC,11,10,62]   [CDAB,2,15,63]    [BCDA,9,21,64]
A = A + AA
B = B + BB
C = C + CC
D = D + DD
end
```

第 5 步,输出结果。最终将 ABCD 组合起来,就构成原消息的摘要。

例 7.3　已知消息为“Beijing610104197503172434”,求该消息的摘要。

'B'=01000010　'e'=01100101　'i'=01101001

'j'=01101010　'n'=01101110　'g'=01100111

'0'=00110000　'1'=00110001　'2'=00110010

'3'=00110011　'4'=00110100　'5'=00110101

'6'=00110110　'7'=00110111　'9'=00111001

消息长度为 200=25×8。

200 的二进制表示为:

“0000000000000000000000000011001000”

M[16]=

01101010　01101001　01100101　01000010
00110110　01100111　01101110　01101001
00110000　00110001　00110000　00110001
00110111　00111001　00110001　00110100
00110001　00110011　00110000　00110101
00110011　00110100　00110010　00110111
00000000　00000000　10000000　00110100
00000000　00000000　00000000　00000000
00000000　00000000　00000000　00000000
00000000　00000000　00000000　00000000
00000000　00000000　00000000　00000000
00000000　00000000　00000000　00000000
00000000　00000000　00000000　00000000

00000000 00000000 00000000 00000000
00000000 00000000 00000000 11001000
00000000 00000000 00000000 00000000

ABCD 赋初值：

A＝“01100111010001010010001100000001”
B＝“11101111110011011010101110001001”
C＝“10011000101110101101110011111110”
D＝“00010000001100100101010001110110”

通过迭代运算得到最终 Hash 值：

00001010 00101010 10101110 11000011 10101101 00101000
11000110 01100101 00011011 00011111 00001100 11011011
00100111 10000100 10111111 11111010

将消息做很小的改动重新计算 Hash 值。

Beijing600104197503172434 对应的 Hash 值是：

01010101 01000011 10001100 11100000 11000011 10001011
10110011 11110001 11010101 00001110 01010110 10011010
10011101 01110001 00000011 11100110

Beijing610304197503172434 对应的 Hash 值是：

11011101 10010000 10000000 10011111 01001011 10101100
11101110 00001001 10101110 01110110 10111010 11100101
11000111 00011001 11111011 00110010

可见，虽然以上各消息只有一位的不同，但消息的摘要却有很大差别，这就是所谓的“雪崩效应”（即使只改变消息中的一位，数字摘要也要有 50％的位发生变化）。

RFC1321 对 MD5 算法也进行了详细的描述，此外 MD5 算法的应用标准或草案还包括以下内容。

RFC1828：IP Authentication using Keyed MD5。

RFC1864：The Content-MD5 Header Field。

RFC2082：RIP-2 MD5 Authentication。

RFC2085：HMAC-MD5 IP Authentication with Replay Prevention。

RFC2385：Protection of BGP Sessions via the TCP MD5 Signature Option。

RFC2403：The Use of HMAC-MD5-96 within ESP and AH。

RFC2537：RSA/MD5 KEYs and SIGs in the Domain Name System (DNS)。

RFC3562：Key Management Considerations for the TCP MD5 Signature Option。

SHA 系列的 Hash 函数是美国标准与技术研究所（NIST）设计的。1993 年 NIST 公布了安全 Hash 算法 SHA-0，SHA-0 被美国政府核准作为标准，即 FIPS180 安全 Hash 标准（Secure Hash Standard ，SHS）。很快 SHA-0 的算法被发现了弱点，1995 年 NIST 公布了 SHA-0 的改进版 SHA-1，即 FIPS180-1SHS。SHA-1 的设计思想同样基于 MD4，因此在很多方面与 MD5 算法有相似之处。SHA-1 对任意长度的明文可以生成 160 位的消息摘要，可与 DSA 一起使用。

SHA-1 对明文的处理和 MD5 相同，第一个填充位为“1”，其余填充位均为“0”，然后将原始明文的真实长度表示为 64 位附加在填充结果后面。填充后明文的长度为 512 的整数倍，填充完毕后，明文按照 512 位分组。

SHA-1 算法的循环次数为明文的分组数，对每一个明文分组的操作有 4 轮，每轮 20 个步骤，一共是 80 个步骤。SHA-1 算法流程，如图 7.4 所示。

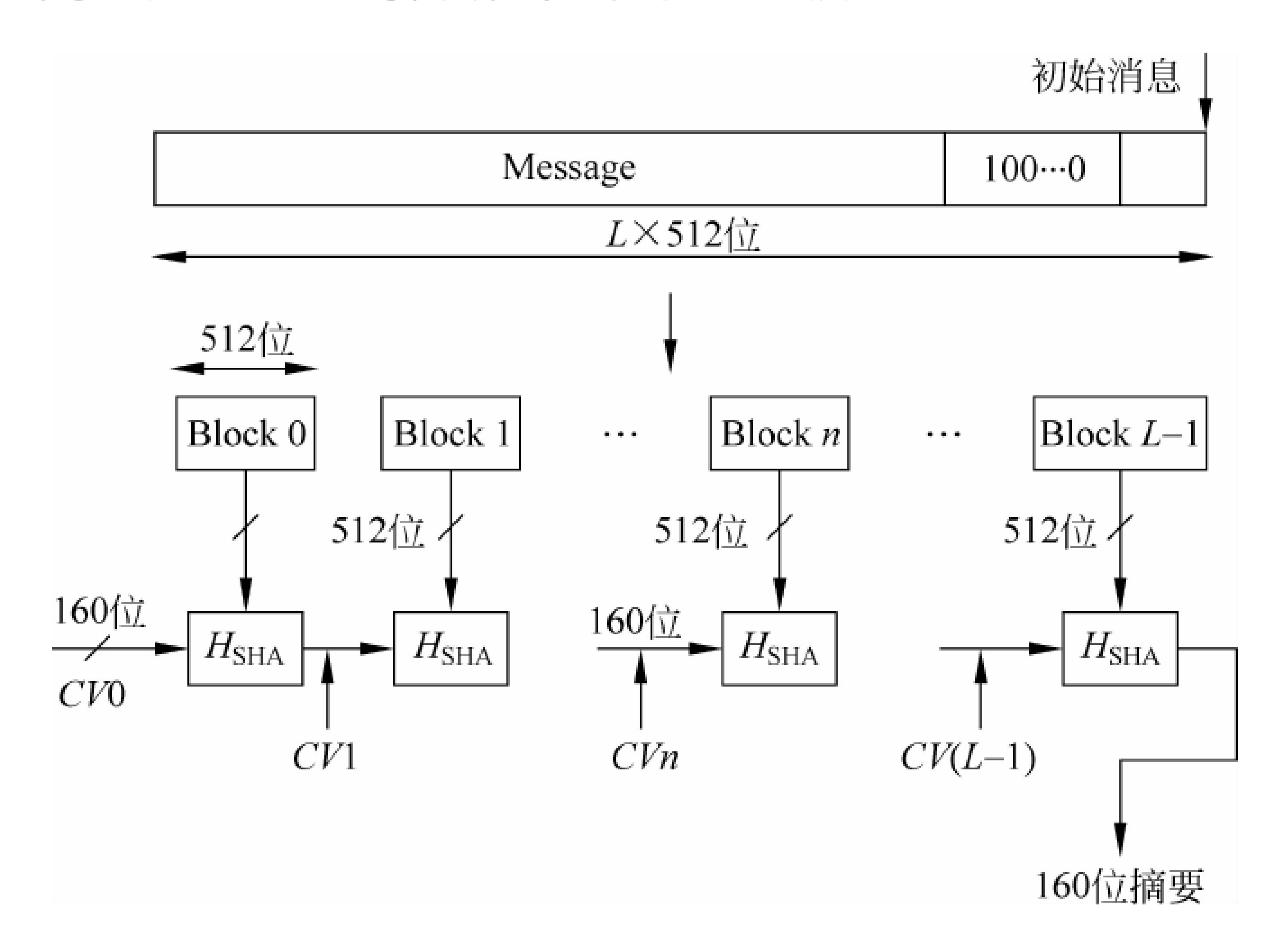


图 7.4　SHA-1 算法流程

在图 7.5 中，每一步对 5 个 Word(32 位)进行操作。这 5 个字的初始值为：

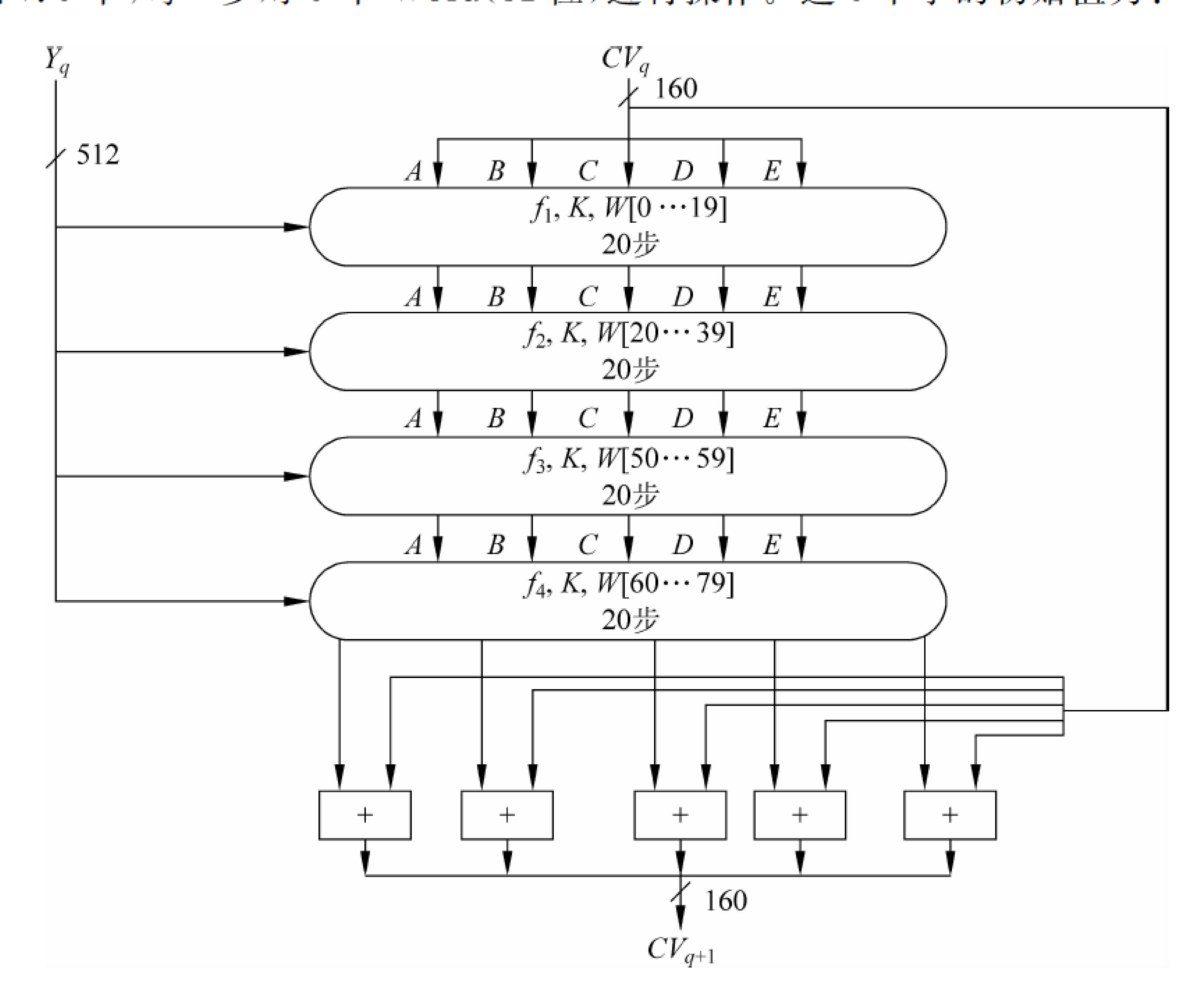


图 7.5　一个分组的运算

$$A = 0x67452301$$
$$B = 0xEFCDAB89$$
$$C = 0x98BADCFE$$
$$D = 0x10325476$$
$$E = 0xC3D2E1F0$$

SHA-1 中使用了一组逻辑函数 F_t（t 表示操作的步骤数，$0 \leqslant t \leqslant 79$）。每个逻辑函数均对 3 个字的变量 B、C、D 进行操作，产生一个字的输出。

逻辑函数 $f_t(B,C,D)$定义如下：

$$f_t(B,C,D) = (B \text{ and } C) \text{ or } (\text{not } (B) \text{ and } D) \quad (0 \leqslant t \leqslant 19)$$
$$f_t(B,C,D) = B \text{ xor } C \text{ xor } D \quad (20 \leqslant t \leqslant 39)$$
$$f_t(B,C,D) = (B \text{ and } C) \text{ or } (B \text{ and } D) \text{ or } (C \text{ and } D) \quad (40 \leqslant t \leqslant 59)$$
$$f_t(B,C,D) = B \text{ xor } C \text{ xor } D \quad (60 \leqslant t \leqslant 79)$$

SHA-1 中还用到一组常数 K_t（t 表示操作的步骤数，$0 \leqslant t \leqslant 79$），每个步骤使用一个。这一组常数的定义为：

$$K_t = 0x\ 5A827999 \quad (0 \leqslant t \leqslant 19)$$
$$K_t = 0x\ 6ED9EBA1 \quad (20 \leqslant t \leqslant 39)$$
$$K_t = 0x\ 8F1BBCDC \quad (40 \leqslant t \leqslant 59)$$
$$K_t = 0x\ CA62C1D6 \quad (60 \leqslant t \leqslant 79)$$

将明文按照规则填充，然后按照 512 位分组为 $M(1)$、$M(2)$、…、$M(L)$，对每个 512 位的明文分组 $M(i)$操作如下（图 7.6）。

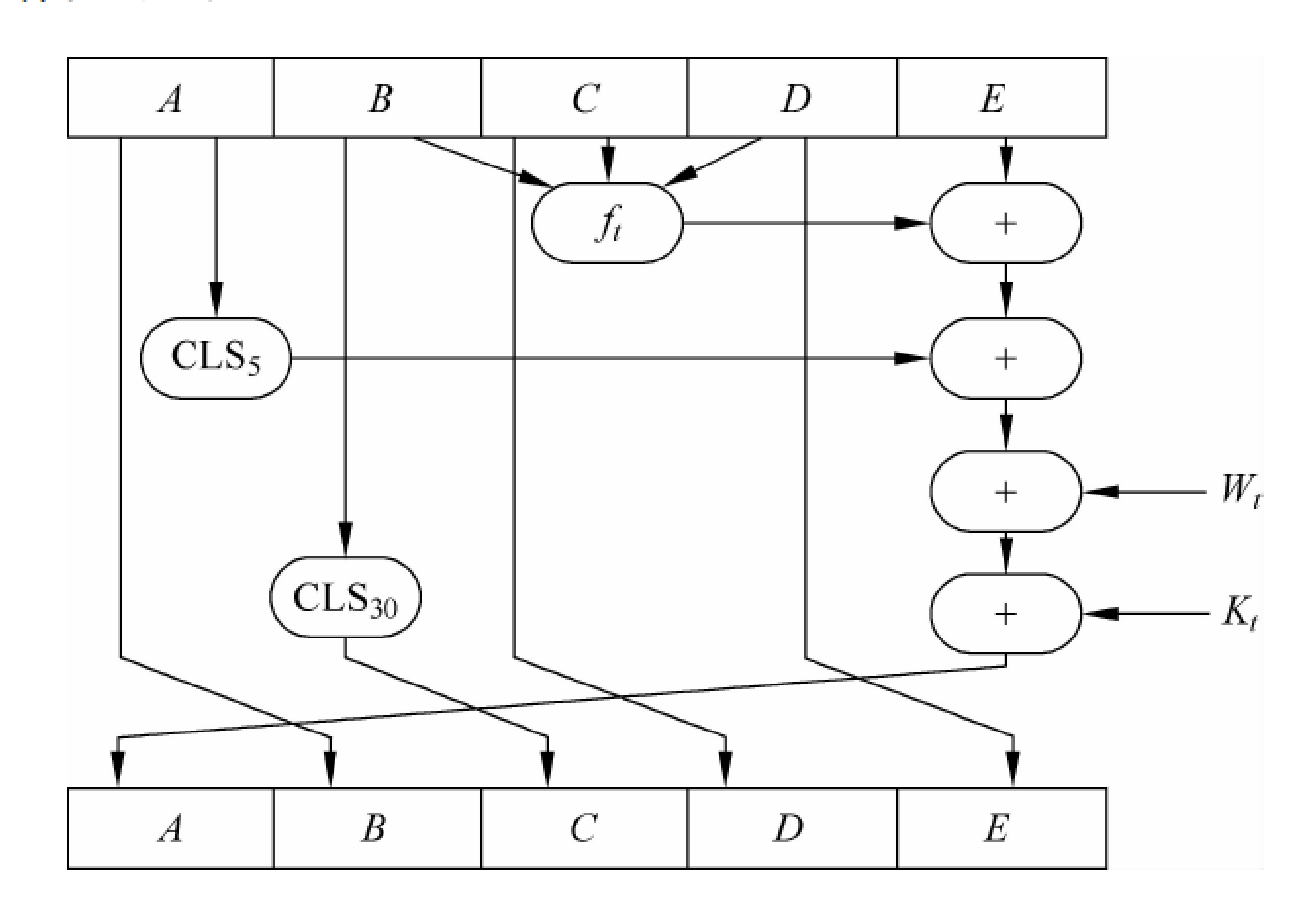


图 7.6　单个回合的运算

(1) 将 512 位的一个明文分组又分成 16 个字的子分组 M_0、M_1、…、M_{15}，M_0是最左边的一个子分组。

(2) 再按照以下法则将 16 个字的子分组变换成 80 个字的分组 W_0、W_1、…、W_{79}。

$$W_t = M_t \quad (0 \leqslant t \leqslant 15)$$
$$W_t = W_{t-3} \text{ xor } W_{t-8} \text{ xor } W_{t-14} \text{ xor } W_{t-16} \quad (16 \leqslant t \leqslant 79)$$

(3) 将 5 个工作变量中的数据复制到另外 5 个记录单元中，令：

$$A = H0$$

$$B = \mathrm{H1}$$
$$C = \mathrm{H2}$$
$$D = \mathrm{H3}$$
$$E = \mathrm{H4}$$

(4) 进行 4 轮共 80 个步骤的操作，t 表示操作的步骤数，$0 \leqslant t \leqslant 79$：

$$\mathrm{TEMP} = A <<< 5 + F_t(B,C,D) + E + W_t + K_t$$
$$E = DD = CC = B <<< 30B = AA = \mathrm{TEMP}$$

(5) 第 4 轮的最后一步完成后，再作运算：

$$\mathrm{H0} = \mathrm{H0} + A$$
$$\mathrm{H1} = \mathrm{H1} + B$$
$$\mathrm{H2} = \mathrm{H2} + C$$
$$\mathrm{H3} = \mathrm{H3} + D$$
$$\mathrm{H4} = \mathrm{H4} + E$$

以上"＋"是指 mod 2^{32} 的加运算。得到的 5 个记录单元中的 H0、H1、H2、H3 和 H4 成为下一个分组处理时的初始值。等最后一个明文分组处理完毕后，5 个工作变量中的数值连接后就成为最终的 Hash 值。

例 7.4 已知消息是"Beijing610104197503172434"，则相应的 SHA-1 值是：

00101110 01011001 11000111 11110110 11001001 00001110
00010010 00011011 01011111 01110010 10000011 11010010
11100011 11111000 10001011 11111111 11101101 11010001
11101111 10010101

将相应的消息仅变动一位："Beijing600104197503172434"，则相应的 SHA-1 值是：

11000000 01011010 10010111 01010001 11001011 01100011
01111110 11101101 00100010 01000110 00111110 01101111
00011100 00010110 10000110 11010110 10110110 11100111
00011001 10011110

NIST 在 2002 年 8 月 1 日发布了 FIPS180-2(SHA-2) 以替换 FIPS180-1(SHA-1)，并附加 3 个可以产生较大长度消息摘要的算法。在 FIPS180-2 中描述的 SHA-1 算法和 FIPS180-1 中描述的 SHA-1 算法相同，但为了与 SHA-256、SHA-384 和 SHA-512 保持一致，也做了一些符号上的改动。

FIPS180-2 中的 4 个算法在明文分组的长度和计算过程中的使用的基本单元数方面有所不同，如表 7.1 所示。

表 7.1 FIPS180-2 中的 4 个算法的比较

算法	消息长度(位)	分组长度(位)	字长(位)	消息摘要长度(位)	安全性(位)
SHA-1	$<2^{64}$	512	32	160	80
SHA-256	$<2^{64}$	512	32	256	128
SHA-384	$<2^{128}$	1024	64	384	192
SHA-512	$<2^{128}$	1024	64	512	256

1. SHA-256

SHA-256 能够对一则长度为 L 的消息 M 产生 Hash 值，$0 \leqslant L < 2^{64}$。

在该算法中，使用了一个由 64 个 32 位单元构成的消息表（Message Schedule），8 个字的工作变量和中间值，最终生成 256 位的消息摘要。

消息表记作 W_0、W_1、…、W_{63}，8 个工作变量记作 a、b、c、d、e、f、g 和 h，中间值记作 $H_0^{(i)}$、$H_1^{(i)}$、…、$H_7^{(i)}$。

$H^{(0)}$ 用指定的数值初始化，运算过程中不断被每个分组处理完成后的中间值 $H^{(i)}$ 替代，最终被最后 $H^{(i)}$ 替代，SHA-256 也使用了两个临时变量 T_1 和 T_2。

1）SHA-256 的预处理

（1）类似前面的方式，按照规则填充消息 M。

（2）将填充后的消息分成 L 个 512 位的分组 $M^{(1)}$、$M^{(2)}$、…、$M^{(L)}$。

（3）用十六进制数初始化 8 个工作变量 $H^{(0)}$：

$$H_0^{(0)} = \text{0x 6a09e667}$$
$$H_1^{(0)} = \text{0x bb67ae85}$$
$$H_2^{(0)} = \text{0x 3c6ef372}$$
$$H_3^{(0)} = \text{0x a54ff53a}$$
$$H_4^{(0)} = \text{0x 510e527f}$$
$$H_5^{(0)} = \text{0x 9b05688c}$$
$$H_6^{(0)} = \text{0x 1f83d9ab}$$
$$H_7^{(0)} = \text{0x 5be0cd19}$$

2）SHA-256 的迭代计算

SHA-256 用函数和常量来计算中间结果和最终 Hash 值，“+”均指 mod 2^{32} 的模加法。

预处理完成后，将每个消息分组分成 16 个字的子分组，再按以下规则变换成 64 个字的子分组，组成一个有 64 字单元的消息表，按次序依次进行处理。

```
For i = 1 to L: {
(1) 计算产生消息表{W_t}:
        W_t = M_t^(i)                                              0≤t≤15
        W_t = σ1^{256}(W_{t-2}) + W_{t-7} + σ0^{256}(W_{t-15}) + W_{t-16}   16≤t≤63
(2) 以第 i-1 圈的计算结果初始化 8 个工作变量 a, b, c, d, e, f, g 和 h:
            a = H_0^(i-1)        b = H_1^(i-1)
            c = H_2^(i-1)        d = H_3^(i-1)
            e = H_4^(i-1)        f = H_5^(i-1)
            g = H_6^(i-1)        h = H_7^(i-1)
(3)
      For t = 0 to 63: {
        T_1 = h + Σ1^{256}(e) + Ch(e,f,g) + K_t^{256} + W_t
        T_2 = Σ0^{256}(a) + Maj(a,b,c)
        h = g
        g = f
        f = e
        e = d + T_1
        d = c
```

```
        c = b
        b = a
        a = T1 + T2
        }
(4) 计算第 i 次迭代的中间值 H(i):
            H0(i) = a + H0(i-1)        H1(i) = b + H1(i-1)
            H2(i) = c + H2(i-1)        H3(i) = d + H3(i-1)
            H4(i) = e + H4(i-1)        H5(i) = f + H5(i-1)
            H6(i) = g + H6(i-1)        H7(i) = h + H7(i-1)
}
```

经过 L 次迭代之后，消息 M 最终的 256 位消息摘要是 8 个工作变量连接后的数据：

$$H_0^{(L)} \parallel H_1^{(L)} \parallel H_2^{(L)} \parallel H_3^{(L)} \parallel H_4^{(L)} \parallel H_5^{(L)} \parallel H_6^{(L)} \parallel H_7^{(L)}$$

2. SHA-512

SHA-512 能够对一则长度为 L 的消息 M 产生 Hash 值，$0 \leqslant L < 2^{128}$。

在该算法中，使用了一个由 80 个 64 位单元构成的消息表，8 个 64 位的工作变量和中间值，最终生成 512 位的消息摘要。

消息表记作 W_0、W_1、…、W_{79}，8 个工作变量分别记作 a、b、c、d、e、f、g 和 h，中间值记作 $H_0^{(i)}$、$H_1^{(i)}$、…、$H_7^{(i)}$。

$H^{(0)}$用给定的数值初始化，运算过程中不断被每个分组的中间值 $H^{(i)}$ 代替，最终被最后 $H^{(L)}$ 代替，SHA-512 也使用了两个临时变量 T_1 和 T_2。

1）SHA-512 的预处理

（1）按照规则填充消息 M。

（2）将填充后的消息分成 L 个 1024 位的分组 $M^{(1)}$、$M^{(2)}$、…、$M^{(L)}$。

（3）用十六进制数初始化 8 个工作变量 $H^{(0)}$：

$H_0^{(0)}$ = 0x 6a09e667 f3bcc908　$H_1^{(0)}$ = 0x bb67ae85 84caa73b

$H_2^{(0)}$ = 0x 3c6ef372 fe94f82b　$H_3^{(0)}$ = 0x a54ff53a 5f1d36f1

$H_4^{(0)}$ = 0x 510e527f ade682d1　$H_5^{(0)}$ = 0x 9b05688c 2b3e6c1f

$H_6^{(0)}$ = 0x 1f83d9ab fb41bd6b　$H_7^{(0)}$ = 0x 5be0cd19 137e2179

2）SHA-512 的迭代计算

SHA-512 的迭代计算用到函数和常量，运算中的“+”都是 mod 2^{64} 的加法。

预处理完成后，将每个消息分组成 16 个 64 位的子分组，再按以下规则变换成 80 个 64 位的子分组，组成一个有 80 个 64 位单元的消息表，按次序依次处理。

```
For i = 1 To N: {
(1) 对预处理后的消息产生消息表{Wt}:
        Wt = Mt(i)                                            0≤t≤15
        Wt = s1{256}(Wt-2) + Wt-7 + s0{256}(Wt-15) + Wt-16    16≤t≤79
(2) 用第 i-1 次迭代的中间值初始化 8 个工作变量 a, b, c, d, e, f, g 和 h:
          a = H0(i-1)    b = H1(i-1)
          c = H2(i-1)    d = H3(i-1)
          e = H4(i-1)    f = H5(i-1)
          g = H6(i-1)    h = H7(i-1)
```

(3)

```
For T = 0 To 79 {
T1 = h + ∑1{256}(e) + Ch(e,f,g) + Kt{256} + Wt
T2 = ∑0{256}(a) + Maj(a,b,c)
h = g
g = f
f = e
e = d + T1
d = c
c = b
b = a
a = T1 + T2
}
```

(4) 计算第 i 次迭代的中间值 $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)} \qquad H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)} \qquad H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)} \qquad H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)} \qquad H_7^{(i)} = h + H_7^{(i-1)}$$

经过 N 次迭代后，消息 M 最终的 512 位的消息摘要是 8 个工作变量连接而成的数据：

$$H_0^{(L)} \parallel H_1^{(L)} \parallel H_2^{(L)} \parallel H_3^{(L)} \parallel H_4^{(L)} \parallel H_5^{(L)} \parallel H_6^{(L)} \parallel H_7^{(L)}$$

3. SHA-384

SHA-384 能够对一则长度为 L 的消息 M 产生 Hash 值，$0 \leqslant L < 2^{128}$。该算法除以下两点外，其余与 SHA-512 基本相同。

(1) 工作变量 $H^{(0)}$ 按照以下值进行初始化：

$$H_0^{(0)} = 0\text{x cbbb9d5d c1059ed8}$$
$$H_1^{(0)} = 0\text{x 629a292a 367cd507}$$
$$H_2^{(0)} = 0\text{x 9159015a 3070dd17}$$
$$H_3^{(0)} = 0\text{x 152fecd8 f70e5939}$$
$$H_4^{(0)} = 0\text{x 67332667 ffc00b31}$$
$$H_5^{(0)} = 0\text{x 8eb44a87 68581511}$$
$$H_6^{(0)} = 0\text{x db0c2e0d 64f98fa7}$$
$$H_7^{(0)} = 0\text{x 47b5481d befa4fa4}$$

(2) 最终得到的 Hash 值是最后一次迭代的中间值 $H^{(L)}$ 最左端的 384 位：

$$H_0^{(L)} \parallel H_1^{(L)} \parallel H_2^{(L)} \parallel H_3^{(L)} \parallel H_4^{(L)} \parallel H_5^{(L)}$$

例 7.5 已知消息是"Beijing610104197503172434"，则相应的 SHA-256 算法的 Hash 函数值是：

01011000 11110011 11100011 10010101 00000000 10010011
00110111 00110110 11010010 00111011 11100101 00000100
10111111 11011110 00110110 00101110 00110000 10101010
00001100 00011110 11000001 11111110 00000111 01100111
00000100 10111110 00000000 10110010 00111111 10011110
11110100 11111101

相应的 SHA-384 算法的 Hash 函数值是：

01110100　00100000　10001111　01010010　11100100　11010100
10100100　11101011　01011101　10000010　11110110　00001001
00001001　01101110　11001010　10010011　00011000　10001111
00101101　11001100　01111000　11000011　10100111　10111101
11110110　10111010　10101001　01101000　00110111　00101000
00001110　01010011　00110011　11010010　10011011　10111010
00000010　10000100　00000110　00100111　01011010　11000010
01011000　00110000　10100011　11101111　10011010　00110111

相应的 SHA-512 算法的 Hash 函数值是：

00111110　10100000　10001011　01100011　10101001　11010001
00110110　10000100　00111001　10101100　10001110　11101010
00011000　10011000　11111000　01011010　01010011　11010100
01111111　10011000　10101101　11111011　11101000　00100010
00010000　10100101　10110111　11010000　00110011　01011000
01101011　10000110　01011110　10110001　11001100　11100101
00000100　01001000　11010011　11001010　00111110　00110010
01111010　10101000　00101101　11110101　10110101　10100001
00111011　00101001　00000111　01001011　11110010　00001101
01011010　10010110　10010110　01100111　00000101　10000010
01001010　01001000　01111001　10011110

7.4　消息认证码

前面提到 Hash 函数的一个作用是配合数字签名的实施。在数字签名之前，需要先对消息进行 Hash 函数运算，压缩成一定的长度。Hash 函数还有另外一个重要的作用就是数据完整性验证。

所谓数据完整性，是指防止数据未经授权的篡改，是信息安全的一项基本要求。为保证接收到的消息与原消息相同，可验证原消息的 Hash 函数值和收到消息的 Hash 函数值是否相同。从前面的例子可以看到，即使仅仅是 1 位的变化，也能使 Hash 函数值变得面目全非。一般来讲，数据完整性的破坏可能有两个原因，一种是传输过程中的差错，另一种是人为蓄意篡改。无论是哪种情况下的数据完整性保护，Hash 函数都是适合的。

验证数据完整性，又称消息认证(Message Authentication)。

前面介绍的 Hash 函数都是无须密码的，但在有些场合下却需要密码。例如，发送方通过公开信道将消息发送给接收方，接收方不仅希望收到消息本身，还要确定消息是否来自合法的发送方。此时，就需要发送方和接收方共享一个密钥，在进行 Hash 运算时使用到这个密钥。带密钥的 Hash 函数是一种消息认证码(Message Authentication Code，MAC)。

在图 7.7 中，MAC＝$Ck(M)$是消息和密钥的函数。虽然带密钥的 Hash 函数可以构造

出 MAC,但反过来,MAC 并不是只能通过 Hash 函数来构造。

通过 Hash 函数来构造 MAC 的关键是如何将密钥加入到 Hash 过程中。

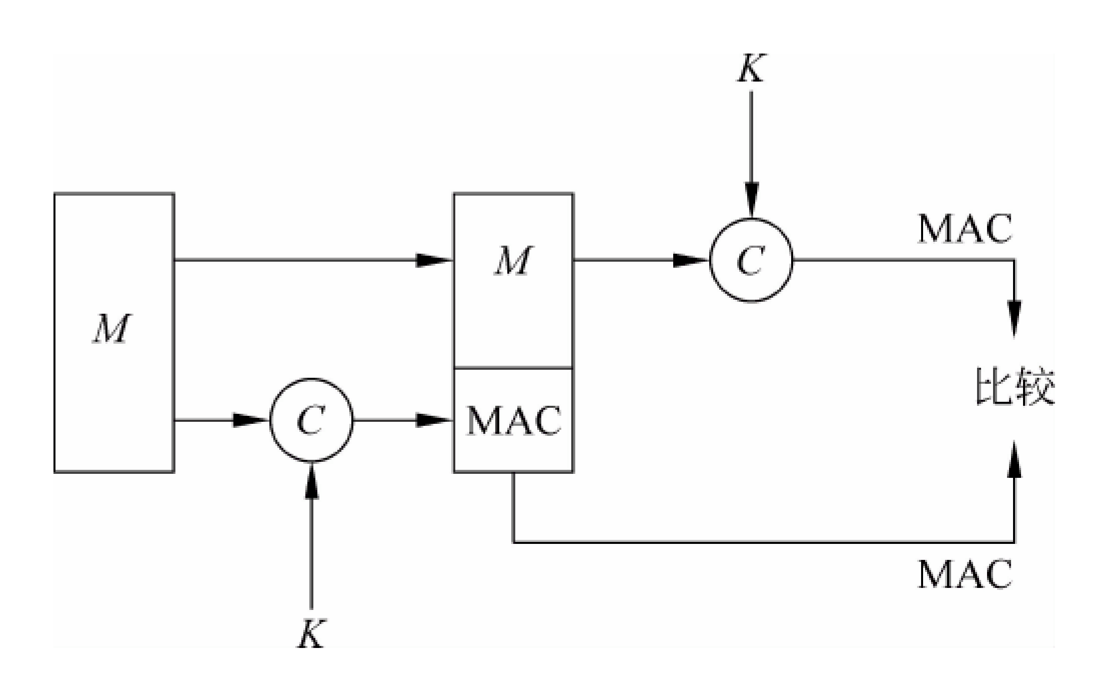


图 7.7 MAC 原理图

可以选取一个具有迭代结构的 Hash 函数来构造,如图 7.8 所示。下面以用 SHA-1 构造的 HMAC 为例来介绍。

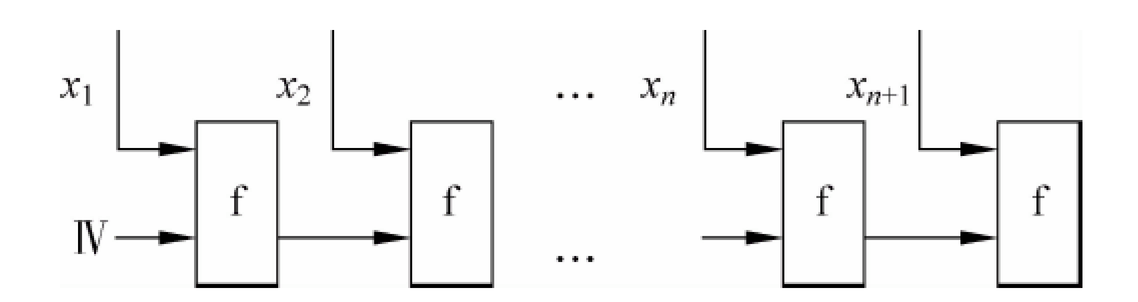


图 7.8 迭代结构的 Hash 函数

设消息 $x=(x_1,x_2,\cdots,x_{L-1})$,其中 x_i 是长度为 b 的块,在 SHA-1 算法中 $b=512$,$i=1,\cdots,L-1$,$L-1$ 是总块数。x_L 也是长度为 b 的二进制串,其中包含了 x 最后不足 b 的部分和整个消息长度的二进制编码以及其他填充位。

IV 代表初始变量,其长度为 q,在 SHA-1 算法中 $q=160$。密钥 k 的长度通常小于或等于 b,设 K'表示 K 用 0 填充到 b 位的结果。Ipad 和 Opad 是 b 位的常量二进制串:

$$\text{Ipad}=3636\cdots36$$

$$\text{Opad}=5\text{C}5\text{C}\cdots5\text{C}$$

那么

$$\text{HMAC}_K(x)=\text{Hash}((K'\oplus\text{Opad})\ \|\ \text{Hash}((K'\oplus\text{Ipad})\ \|\ x))$$

作为一种消息认证码,HMAC 简单而高效,目前已经取代了 RFC1828 成为 IPSec 协议中的认证算法。

另一类主流的 MAC 构造方法是基于分组密码的 CBC (Cipher Block Chaining)模式构造的 MAC,如图 7.9 所示。

设消息 $M=D_1\ \|\ D_2\ \|\ \cdots|\ D_N$,使用对称分组加密体制(如 DES),分组长度 m(在 DES 中 $m=64$)。不妨设 O_0=初始值,则:

$$O_i = E_{D_i}(O_{i-1})$$

最终的 CBC-MAC 为:

$$G = O_N$$

CBC-MAC 已被很多国际组织和机构定为标准,如标准 ANSI X9.9、FIPS-PUB113 和 ISO/IEC9797-1。

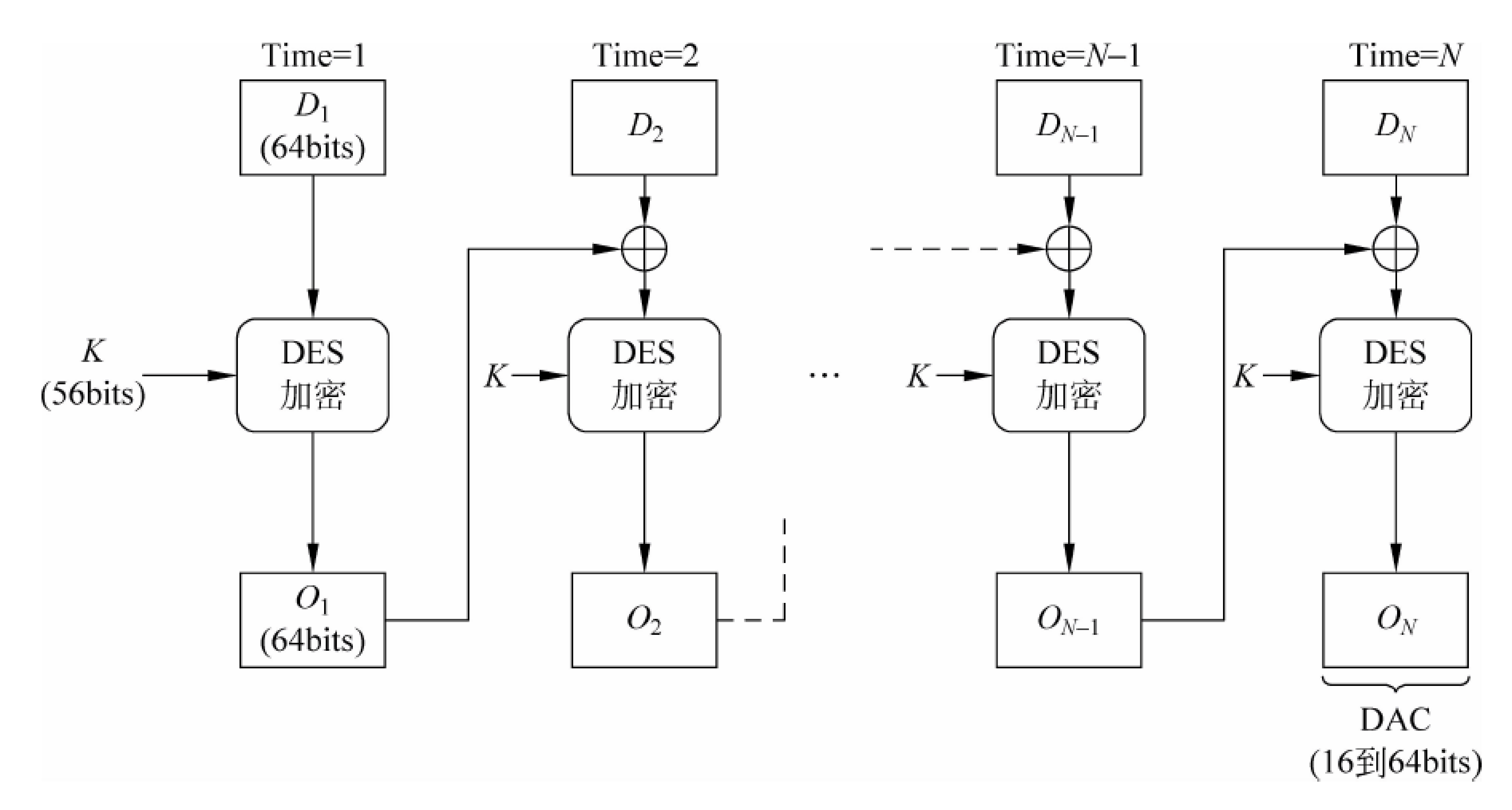


图 7.9　CBC-MAC 示意图

接下来讨论 MAC 抗穷举攻击的安全性。对 MAC 的穷举攻击是指给定任意长度的消息和相应的 MAC 求可能的密钥。

设 MAC 的长度为 n 位，于是将有 2^n 个可能的 MAC，消息的数量一般应大于 MAC 的数量，设为 $N(N \geqslant 2^n)$，密钥的长度为 k，表明有 2^k 个可能的密钥。

假设攻击者已经获得消息的明文和相应的 MAC，即(M_i, MAC_i)。

第 1 轮：给定 M_1 和 $\mathrm{MAC}_1 = C_K(M_1)$，对所有 2^k 个可能的密钥计算：

$$\mathrm{MAC}_1 = C_{Ki}(M_1) \quad (i = 1, 2, \cdots, 2^k)$$

发现匹配的数量约为 2^{k-n}，得到 2^{k-n} 个可能的密钥。

第 2 轮：给定 M_2 和 $\mathrm{MAC}_2 = C_K(M_2)$，对前面得到的 2^{k-n} 个可能的密钥计算：

$$\mathrm{MAC}_2 = C_{Ki}(M_2) \quad (i = 1, 2, \cdots, 2^{k-n})$$

发现匹配的数量约为 2^{k-2n}，得到 2^{k-2n} 个可能的密钥。

……

如果 $k > n$（设 $k = a \times n$），则这种攻击要得到密钥，需要进行 a 轮运算，即一共需要 $a \times 2^k$ 次以上的 MAC 运算。

如果 $k \leqslant n$，则第 1 轮就可以产生一个唯一对应的密钥，但仍然可能有两个或两个以上的密钥产生这一消息 MAC 对（当 $k < n$ 时），这时攻击者需对一个新的(M_2, MAC_2)进行相同的计算，因此也需要大于 2^k 次的 MAC 运算。

综上所述，无论 $k > n$ 还是 $k \leqslant n$，攻击者要获得密钥都至少需要 2^k 次的 MAC 运算。由此可见，攻击者企图发现 MAC 的密钥不小于甚至大于对同样长度的解密密钥的攻击，对 MAC 的穷举攻击意义不大。

7.5　习　　题

1. 用 Chaum-van Heijst-Pfitzmann Hash 函数对消息 $x = (23, 37)$ 进行数字摘要，具体参数为 $p = 1019, q = 509, \alpha = 16, \beta = 435$。

2. 将 10 个球随机扔进 1000 个筐子中，试求不出现同一个篮子中有两个及两个以上球的概率。

3. 用 MD5 对以下消息进行摘要：

(1)“beijing2008”

(2)“ChaumvanHeijstPfitzmannHashFunction”

4. 简述 Hash 函数与消息验证码的关系。

5. 给定 Hash 函数 h 和任意给定的消息 m，寻找一个 m'，$m' \neq m$，使得 $h(m')=h(m)$ 的问题称为________问题。

6. 在 MD5 算法中，将消息划分成________位的消息块进行处理，最终形成________位的信息摘要。

7. 在 MD5 算法中，追加的长度信息由________位表示，被追加到已补的信息后，如果原消息长度超过________位，只使用________即可。

8. SHA-1 算法的循环次数为明文的分组数，对每一个明文分组的操作有________轮，每轮________个步骤。

9. SHA-384 算法的分组长度是________位，字长是________位，消息摘要长度是________位。

第 8 章　密钥管理与公钥基础设施

无论是对称密码系统还是公钥密码系统，系统安全的焦点是密钥。本章主要介绍两方面的技术：

（1）密钥的分发和协商方面的技术；

（2）公钥基础设施的概念及其相关技术。

8.1　密钥预分发方案

应该说，公钥密码系统比对称密码系统更适于加密通信，因为在公钥密码系统中通信双方无须共享同一个密钥，也就省去了传输密钥的过程，安全性更高。但是，实际上从速度方面讲公钥密码系统却要差一些。因此，在具体应用中，仍然是通过对称密码系统来实现加密信息或消息，而公钥密码系统则可以用来产生对称密码系统的密钥，或加密少量的重要信息。

因此公钥密码系统淘汰对称密码系统是不切实际的，然而只要使用对称密码系统，就存在共享密钥分发与协商的问题。密钥分发是指通信一方将密钥分发给其他几个通信参与方；密钥协商则是指通信双方在一个公共通道上协商共享密钥。

密钥分发通常有三方或三方以上的参与者，那么需要有一个密钥分发中心（Key Distributed Center，KDC）值得大家信赖，并负责为大家准备密钥。根据通信过程中密钥分发的时机来分，密钥分发一般分为预分发和在线分发两种。

事先分发好的密钥，通常有效期较长，并且能够在有效期内反复使用。接下来介绍的 DH 密钥分发方案是建立在离散对数问题基础上的一种密钥预分发方案。

DH 密钥分发方案：公开一个素数 p，取 $\alpha \in Z_p^*$ 。

V 计算：

$$K_{U,V} = \alpha^{a_U a_V} \bmod p = b_U^{a_V} \bmod p$$

其中，b_U 从 U 的证书中得到，a_V 是自己的私钥。

U 计算：

$$K_{U,V} = \alpha^{a_U a_V} \bmod p = b_V^{a_U} \bmod p$$

其中，b_V 从 V 的证书中得到，a_U 是自己的私钥。

例 8.1　在 U、V 之间实施 DH 密钥分发方案。

公开一个素数 $p=25\ 307$ 和相应的生成元 $\alpha=2$。

U 选择私钥 $a_U=3578$，并计算出公钥：

$$b_U = 2^{3578} \bmod 25\ 307 = 6113$$

将 b_U 放进自己的证书中。

V 选择私钥 $a_U=19\ 956$，并计算出公钥：

$$b_V = 2^{19\,956} \bmod 25\,307 = 7984$$

将 b_V 放进自己的证书中。

V 计算：

$$K_{U,V} = 6113^{19\,956} \bmod 25\,307 = 3694$$

b_U 可从 U 的证书中得到。

U 计算：

$$K_{U,V} = 7984^{3578} \bmod 25\,307 = 3694$$

b_V 可从 V 的证书中得到。

DH 密钥预分发方案是一种设计巧妙并且安全性很高的方案。

但是，如果 n 个通信方需要相互通信，那么 KDC 需要维护 $n(n-1)/2$ 个不同的密钥，因为任意两个通信方之间的密钥应该不同。这些密钥的维护量非常大，需要分配很大的空间，尤其当考虑到密钥的定期更新时，问题就更加突出。

针对这个容量问题，下面来看 Blom 密钥预分发方案。

Blom 密钥预分发方案：公开素数 p，对每个用户 U，$r_U \in Z_p$ 也被公开，对于不同的用户 r_U 互不相同。

KDC 选择 3 个随机数 a、b、$c \in Z_p$（3 个随机数不必各不相同），构成多项式

$$f(x,y) = a + b(x+y) + cxy \bmod p$$

对于每一个用户 U，KDC 计算出：

$$g_U(x) = f(x, r_U) \bmod p$$

然后将 $g_U(x)$ 通过安全通道发送给 U，$g_U(x)$ 可标识成：

$$g_U(x) = a_U + b_U x$$

如果 U 和 V 需要通信，他们可以使用密钥：

$$K_{U,V} = K_{V,U} = f(r_U, r_V) = a + b(r_U + r_V) + c(r_U r_V) \bmod p$$

其中，U 获得密钥 $K_{U,V}$ 的办法是：

$$g_U(r_V) = a_U + b_U r_V$$

V 获得密钥 $K_{U,V}$ 的办法是：

$$g_U(r_U) = a_U + b_U r_U$$

例 8.2 在 U、V、W 之间实施 Blom 密钥分发方案。

(1) 公开素数 $p=31$，对用户 U、V、W，选择：

$$r_U = 13, \quad r_V = 11, \quad r_W = 5$$

(2) KDC 选择 3 个随机数：

$$a = 5, \quad b = 6, \quad c = 7$$

构成多项式选择

$$f(x,y) = 5 + 6(x+y) + 7xy$$

(3) 对于用户 U、V、W，KDC 分别计算出：

$$g_U(x) = f(x, 13) \bmod 31$$

$$g_V(x) = f(x, 11) \bmod 31$$

$$g_W(x) = f(x, 5) \bmod 31$$

也可写成：

$$g_U(x) = a_U + b_U x = 21 + 4x$$
$$g_V(x) = a_V + b_V x = 9 + 21x$$
$$g_W(x) = a_W + b_W x = 29 + 10x$$

然后将 $g_U(x)$、$g_V(x)$、$g_W(x)$分别通过安全通道发送给 U、V 和 W。

(4) 如果 U 和 V 需要通信，他们可以使用密钥：

$$\begin{aligned} K_{U,V} = K_{V,U} &= f(r_U, r_V) \\ &= a + b(r_U + r_V) + c(r_U r_V) \bmod p \\ &= 5 + 6 \times (13 + 11) + 7 \times 13 \times 11 \bmod 3 \\ &= 3 \end{aligned}$$

其中，U 获得密钥 $K_{U,V}$的办法是：

$$g_U(r_V) = a_U + b_U r_V = 21 + 4 \times 11 \pmod{31} = 3$$

V 获得密钥 $K_{U,V}$的办法是：

$$g_V(r_U) = a_V + b_V r_U = 9 + 21 \times 13 \pmod{31} = 3$$

同理，如果 U 和 W 需要通信，他们都可以计算相互间的通信密钥：

$$K_{U,W} = K_{W,U} = 10$$

如果 V 和 W 需要通信，他们也都可以计算相互间的通信密钥：

$$K_{V,W} = K_{W,V} = 21$$

图 8.1 是 Blom 密钥预分发方案的示意图。

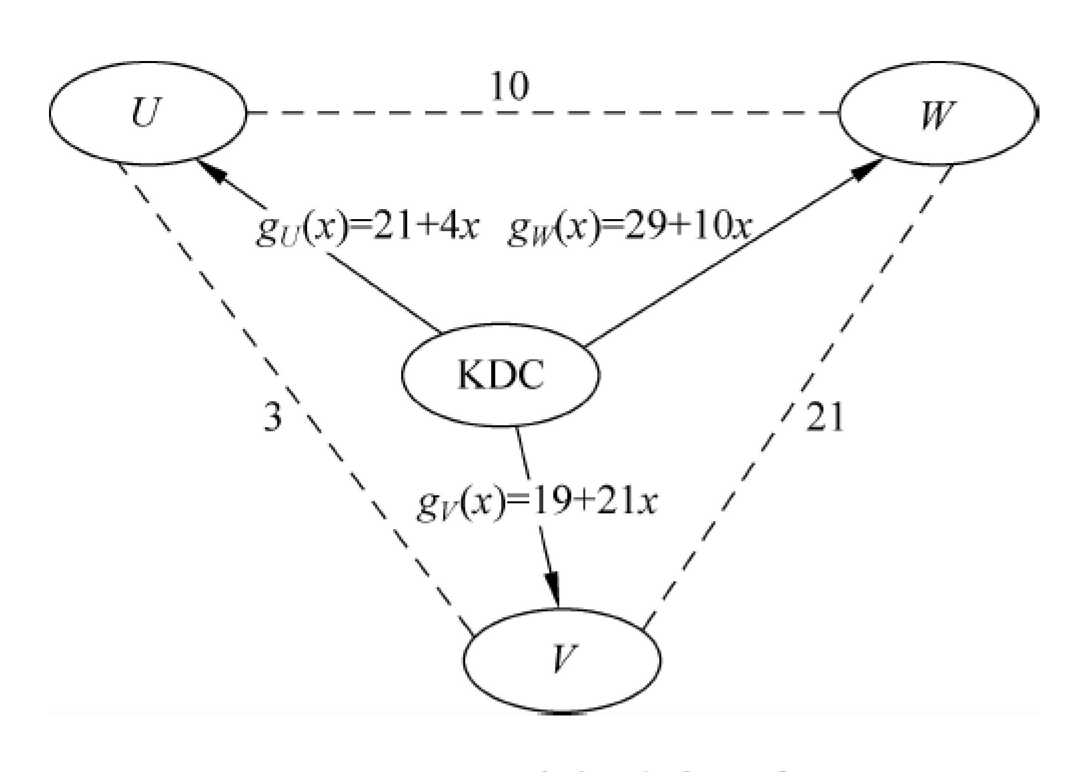


图 8.1　Blom 密钥分发示意图

8.2　密钥在线分发方案

假设通信各方已各自拥有自己的密钥，而该密钥除了自己和 KDC 之外再无人知晓。当通信双方需要通信时，比较成熟的做法是在现有条件下，产生一个新的临时的相互通信的密钥，这个密钥也称为会话密钥(Session Key)。

在线式密钥分配协议每次产生的会话密钥是随机的，也就是说密钥随时在更新，避免了由于一个密钥使用过久而存在的密钥泄露隐患，从而提高了保密通信的安全性。

Needham_Schroeder 密钥分发协议：

$U \to KDC$： $ID(U) \| ID(V) \| N_1$

$KDC \to U$： $E_{KU}[K_S \| ID(V)] \| N_1 \| E_{KV}[K_S \| ID(U)]$

$U \to V$： $E_{KV}[K_S \| ID(U)]$

$V \to U$： $E_{KS}(N_2)$

$U \to V$： $E_{KS}(f(N_2))$

其中，ID(U)和 ID(V)表示 U 和 V 的身份信息；K_U、K_V是 U 和 V 的私钥，K_S是需要在线分发的会话密钥；N_1和 N_2分别是两个随机数；$f(N_2)$是事先约定的一个函数，为简单起见，一般取 $f(N_2)= N_2-1$。

假定攻击者已知一个旧会话密钥，他可以从第(3)步开始假冒 A 并重放。

$P(U) \to V$： $E_{KV}[K_S \| ID(U)]$

$V \to P(U)$： $E_{KS}(N_3)$

$P(U) \to V$： $E_{KS}(f(N_3))$

可见，不能有效地防范重放攻击是 Needham_Schroeder 密钥分发协议的一个缺陷。为此，有以下的改进方案。

Denning 方案：

$U \to KDC$： $ID(U) \| ID(V)$

$KDC \to U$： $E_{KU}[K_S \| ID(V)] \| T \| E_{KV}[K_S \| ID(U) \| T]$

$U \to V$： $E_{KV}[K_S \| ID(U) \| T]$

$V \to U$： $E_{KS}[N_1]$

$U \to V$： $E_{KS}[f(N_1)]$

U 和 V 通过下式来验证及时性：

$$|Clock - T| < \Delta t_1 + \Delta t_2$$

其中，Δt_1是 KDC 与本地的时钟差；Δt_2是网络延时。

但是，显然 Denning 方案对时钟过于依赖。

基于 Needham_Schroeder 密钥分发协议还有很多其他的改进方案，其中最有名的莫过于 Kerboros 协议。

Kerboros 的本意是希腊神话中地狱之门的守护者，该守护者是一个有两个头的怪物。在网络通信模型中，Kerboros 的角色实际上正是 KDC。

Kerberos 协议是美国麻省理工学院在 20 世纪 80 年代中期开发的雅典娜(Athena)项目中的一个部分。Kerberos 协议可以既是一种在线密钥分发系统，也可以看成是一种身份验证系统。

作为在线密钥分发系统，Kerberos 协议采纳了 Denning 和 Sacco 对 Needham_Schroeder 协议的改进建议，在身份验证协议中引入了时间戳。

作为身份验证系统，Kerberos 使得网络服务器可以验证网络用户的身份，无须用户在网络上明文传递密钥。

Kerberos 的最初版本是版本 3，仅供内部使用，而版本 4 是一个投入实际应用的系统。

Kerboros 密钥分发协议：

(1) U 向 KDC 申请一个密钥用于与 V 通信；

(2) KDC 选择一个随机密钥 K_S，加上一个时间戳 T 和一个存活期 L；

(3) KDC 计算出：

$$m_1 = e_{K_U}(K_S, \mathrm{ID}(V), T, L)$$

$$m_2 = e_{K_V}(K_S, \mathrm{ID}(V), T, L)$$

并将 m_1、m_2 发送给 U；

(4) U 使用 d_{K_U} 从 m_1 中计算出 K_S、T、L 和 $\mathrm{ID}(V)$，然后计算出：

$$m_3 = e_{K_S}(\mathrm{ID}(U), T)$$

并将 m_3 和 m_2 发送给 V；

(5) V 使用 d_{K_V} 从 m_2 中计算出 K、T、L 和 $\mathrm{ID}(U)$，然后用 d_{K_S} 从 m_3 中计算出 T 和 $\mathrm{ID}(U)$，检查两个 T 和 $\mathrm{ID}(U)$ 是否一致；

(6) 如果两个 T 和 $\mathrm{ID}(U)$ 一致，V 计算出：

$$m_4 = e_K(T+1)$$

将它发送给 U；

(7) U 使用 e_K 从 m_4 中验证是否为 $T+1$。

图 8.2 描述了 Kerboros 密钥分发的过程。

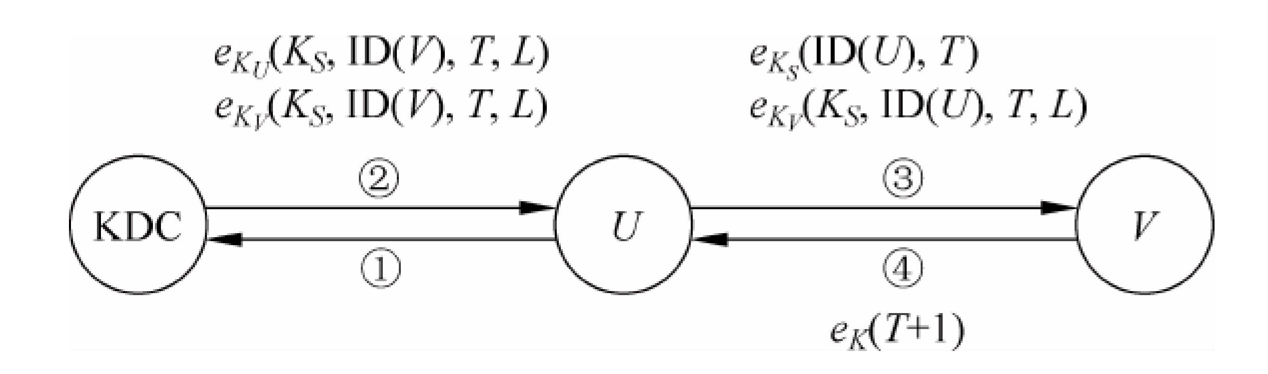


图 8.2　Kerboros 密钥分发协议示意图

① 表示 U 向 KDC 申请密钥；② 表示 KDC 将 m_1 和 m_2 发送给 U；

③ 表示 U 产生 m_3 连同 m_2 发送给 V；④ 表示 V 产生 m_4 发送给 U。

从第③步可以看出 Kerboros 协议中，Kerboros(KDC)是一个双重角色，既是认证服务器，也是授权服务器。这正是为什么给 KDC 起名为 Kerboros(双头怪物)的原因。

Kerboros 沿用至今，目前的成熟版本是 v5。v5 相对与 v4 有以下几项改进：

(1) 传送的所有消息都通过 CBC 模式进行加密；

(2) 使用“重演缓冲区”手段提高避免重放攻击的效率；

(3) 减少了一次没有必要的、重复的加密。

8.3　密钥协商方案

8.2 节中的密钥预分发和在线密钥分发有一个共同的特点，就是需要密钥分发中心(KDC)的参与。密钥协商则不需要 KDC 的参与，通过通信双方的互操作形成一个共享的密钥是基本目标，这一目标通常需采用公钥密码体制来实现。

首先要提到的是 DH 密钥协商方案，DH 密钥协商方案早在 1976 年就已出现。

DH 密钥协商方案与 DH 密钥分发方案非常类似，主要差别在于 a_U 和 a_V 值不是一个固定值，而是在每次会话前随机产生。

DH 密钥协商方案：公开一个素数 p，取 $\alpha \in Z_p^*$。

U 随机选取一个 a_U，然后计算：

$$b_U = \alpha^{a_U} \bmod p$$

并将 b_U 发送给 V。

同样，V 随机选取一个 a_V，然后计算：

$$b_V = \alpha^{a_V} \bmod p$$

并将 b_V 发送给 U。

U 通过计算：

$$K_{U,V} = b_V^{a_U} \bmod p$$

得到共享密钥。

V 通过计算：

$$K_{U,V} = b_U^{a_V} \bmod p$$

得到共享密钥。

然而这种密钥分发方案很脆弱，难以经受中间人(Intruder-in-the-middle)攻击的考验。

图 8.3 说明了 DH 密钥分发方案的中间人攻击示意图。

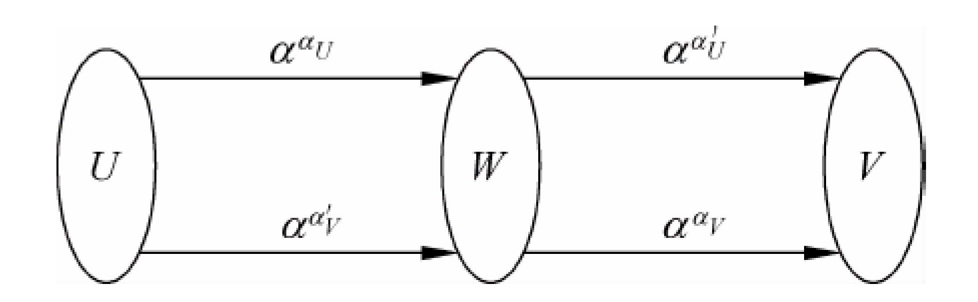


图 8.3 DH 密钥分发方案的中间人攻击

在图 8.3 中，W 与 U 的通信中伪装成 V，与 V 的通信中伪装成 U，而 U 和 V 对这种伪装却全然不知。

不难发现，之所以会遭到中间人攻击，是因为 U 和 V 之间没有进行身份信息的交流。

下面的 STS 密钥协商协议对 DH 密钥分发方案做了改进。

STS 密钥协商协议(简单点到点协议)：

(1) U 选择一个随机数 a_U，$0 \leqslant a_U \leqslant p-2$；

(2) U 计算：

$$\alpha^{a_U} \bmod p$$

并发送给 V；

(3) V 选择一个随机数 a_V，$0 \leqslant a_V \leqslant p-2$；

(4) V 先计算：

$$\alpha^{a_V} \bmod p$$

然后计算：

$$K = (\alpha^{a_U})^{a_V} \bmod p$$

$$y_V = sig_V(\alpha^{a_V}, \alpha^{a_U})$$

(5) V 将 $C(V)$、α^{a_V}、y_V 发给 U，其中：

$$C(V)=(\mathrm{ID}(V),ver_V,sig_{\mathrm{KDC}}(\mathrm{ID}(V),ver_V))$$

(6) U 计算出：

$$K=(\alpha^{a_V})^{a_U} \bmod p$$

并用 ver_V 验证 y_V，用 ver_{KDC}验证 $C(V)$；

(7) U 计算：

$$y_U=sig_U(\alpha^{a_V},\alpha^{a_U})$$

并发送$(C(U),y_U)$给 V；

(8) V 用 ver_U 验证 y_U，用 ver_{KDC}验证 $C(U)$。

图 8.4 说明了 STS 密钥协商协议中的信息传送示意图。

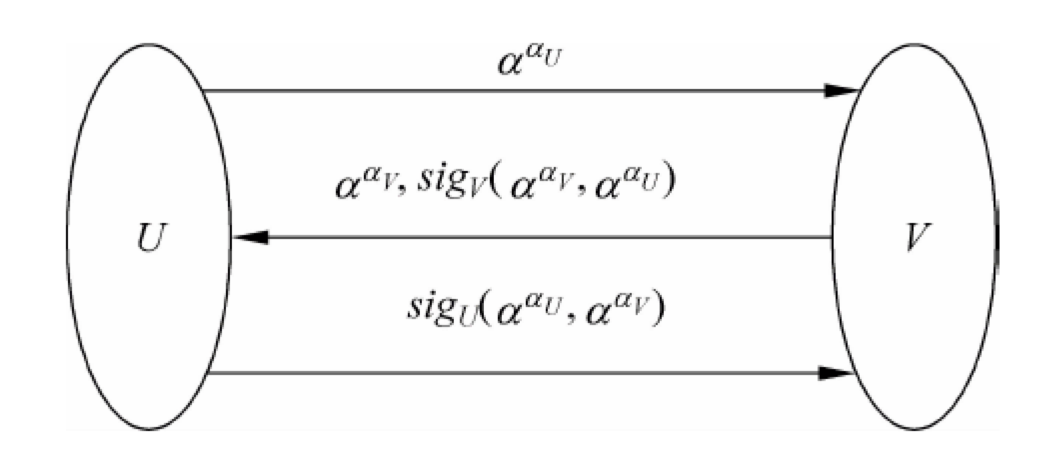


图 8.4　STS 密钥协商示意图

再来考虑中间人攻击，攻击者 W 介入 U 和 V 之间的密钥协商，如图 8.5 所示。

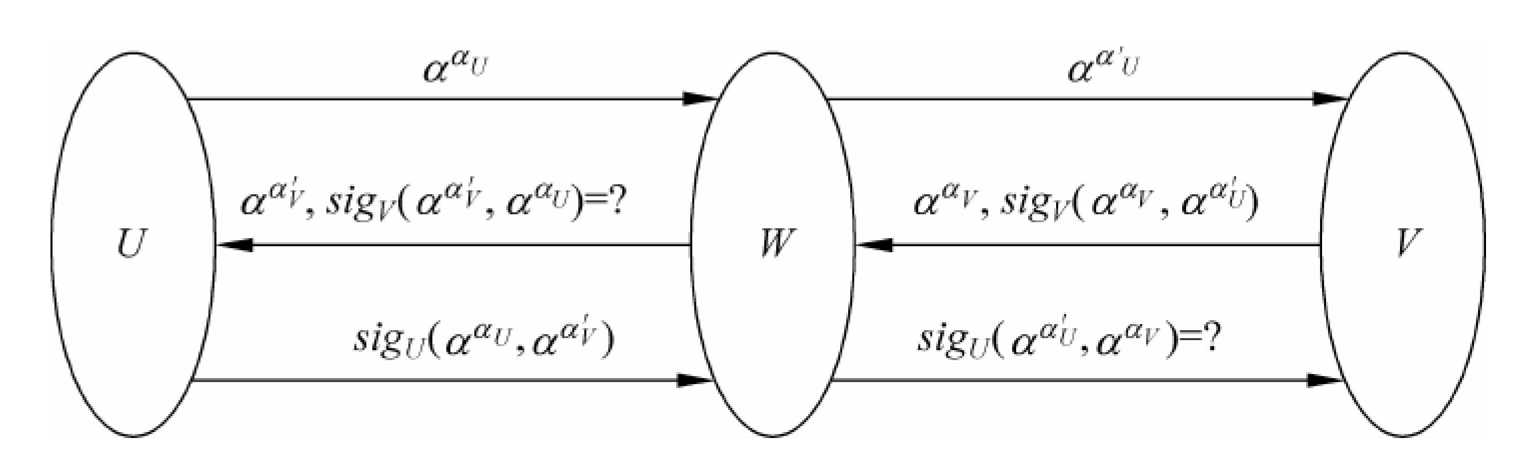


图 8.5　STS 的安全性

攻击者从 U 处截获信息 $\alpha^{a_U} \bmod p$，但是没有把它转交给 V，而是把另外一个数 $\alpha^{a'_U} \bmod p$ 转交给 V。

V 以为 $\alpha^{a'_U} \bmod p$ 是从 U 发出的，于是返回信息 $C(V)$、α^{a_V}、y_V。

W 截获 $C(V)$、α^{a_V}、y_V，W 希望伪造一个信息来冒充 V，但是因为 W 没有 V 的私钥，所以无法伪造 V 的签名；即使 W 能够伪造 V 的签名，U 会再发送 $sig_U(\alpha^{a_v},\alpha^{a'_V})$给 V。

W 截获 $sig_U(\alpha^{a_v},\alpha^{a'_V})$后，希望冒充 U，但是由于 W 没有 U 的私钥，所以无法伪造 U 的签名，很难获得 V 的信任。

综上所述，除非攻击者 W 既能伪造 U 的签名，又能伪造 V 的签名，否则 STS 是能够防止中间人攻击的。

除了 STS 密钥协商协议，Matsumoto、Takashima 和 Imai 也在 DH 方案的基础上构造了一个有趣的密钥协商协议——MTI 密钥协商协议。这个协议有以下两个特点：

(1) U 和 V 不需要进行数字签名；

(2) U 和 V 的信息交换只需两次即可。

MTI 密钥协商协议：

(1) U 的私钥取为 a_U，相应的公钥为：

$$b_U = \alpha^{a_U} \bmod p$$

(2) 设 V 的私钥为 a_V，相应的公钥为：

$$b_V = \alpha^{a_V} \bmod p$$

(3) U 选择一个随机数 r_U，$0 \leqslant r_U \leqslant p-2$，并计算：

$$s_U = \alpha^{r_U} \bmod p$$

(4) U 将 $(C(U), s_U)$ 发给 V，这里的 $C(U)$ 表示 U 的证书；

(5) V 选择一个随机数 r_V，$0 \leqslant r_V \leqslant p-2$，并计算：

$$s_V = \alpha^{r_V} \bmod p$$

(6) V 将 $(C(V), s_V)$ 发给 U；

(7) U 计算出：

$$K = s_V^{a_U} b_V^{r_U} \bmod p$$

其中，b_V 是从 $C(V)$ 中获得；

(8) V 计算出：

$$K = s_U^{a_V} b_U^{r_V} \bmod p$$

其中，b_U 是从 $C(U)$ 中获得。

例 8.3 使用 MTI 密钥协商协议实现 U 和 V 之间的密钥协商。已知 $p = 27\,803$，$\alpha = 5$。

U 的私钥取为：

$$a_U = 21\,131$$

相应的公钥为：

$$b_U = 5^{21\,131} \bmod 27\,803 = 21\,420$$

设 V 的私钥为：

$$a_V = 17\,555$$

相应的公钥为：

$$b_V = 5^{17\,555} \bmod 27\,803 = 17\,100$$

U 选择一个随机数：

$$r_U = 169$$

并计算：

$$s_U = 5^{169} \bmod 27\,803 = 6268$$

U 将 $(C(U), s_U)$ 发给 V，$C(U)$ 中包含了 U 的公钥；

V 选择一个随机数：

$$r_V = 23\,456$$

并计算：

$$s_V = 5^{23\,456} \bmod 27\,803 = 26\,759$$

V 将 $(C(V), s_V)$ 发给 U，$C(V)$ 中包含 V 的公钥；

U 计算出：

$$K = 26\,759^{21\,131} \times 17\,100^{169} \bmod 25\,307 = 21\,600$$

其中,17 100 是从 $C(V)$中获得的 V 的公钥 b_V;

V 计算出:

$$K = 6268^{17\,555} \times 21\,420^{23\,456} \bmod 25\,307 = 21\,600$$

其中,21 420 是从 $C(U)$中获得的 U 的公钥 b_U。

在 MTI 密钥协商协议中,虽然数据交互只有两次,但是由于相互间交换了各自的证书信息,加上攻击者无法获得 U 和 V 的私钥,因此该协议能够抵御中间人攻击。

Girault 密钥协商协议是一种基于自验证公钥的密钥协商协议。与前面的密钥协商协议不同,Girault 密钥协商协议在保证能抵御中间人攻击的前提下,不使用证书。

Girault 密钥协商协议:包括两个过程,第一个过程是自验证密钥的生成过程;第二个过程是密钥协商过程。

(1) 自验证密钥的生成过程如下:

U 选择一个随机数 a_U,计算:

$$b_U = \alpha^{a_U} \bmod n$$

U 将 a_U和 b_U发给 KDC;

KDC 计算出:

$$p_U = (b_U - \mathrm{ID}(U))^d \bmod n$$

KDC 将 p_U发送给 U。

(2) 密钥协商过程如下:

U 选择一个随机数 r_U,计算:

$$s_U = \alpha^{r_U} \bmod n$$

U 将 $\mathrm{ID}(U)$,p_U,s_U发送给 V;

V 选择一个随机数 r_V,计算:

$$s_V = \alpha^{r_V} \bmod n$$

V 将 $\mathrm{ID}(V)$、p_V、s_V发送给 V;

U 计算出:

$$K = s_V^{a_U} (p_V^e + \mathrm{ID}(V))^{r_U} \bmod n$$

V 计算出:

$$K = s_U^{a_V} (p_U^e + \mathrm{ID}(U))^{r_V} \bmod n$$

例 8.4　在 Girault 密钥协商协议中,取 $p=977$,$q=1117$,则:

$$n = pq = 977 \times 1117 = 1\,091\,309$$

$$\varphi(n) = (977 - 1) \times (1117 - 1) = 1\,089\,216$$

取 $\alpha=17$,设 KDC 选取 RSA 加密密钥:

$$e = 109\,981$$

相应的解密密钥是:

$$d = 9973$$

设 $\mathrm{ID}(U)$为 200 409,且 U 选择:

$$a_U = 197\,503$$

则:

$$b_U = 5^{197\,503} \bmod 1\,091\,309 = 690\,191$$

$$p_U = (690\ 191 - 200\ 409)^{9973} \bmod 1\ 091\ 309 = 150\ 608$$

设 ID(V)为 200 510，且 V 选择：

$$a_V = 197\ 702$$

则：

$$b_V = 5^{197\ 702} \bmod 1\ 091\ 309 = 227\ 592$$

$$p_V = (227\ 592 - 200\ 510)^{9973} \bmod 1\ 091\ 309 = 659\ 617$$

前面的过程产生了自验证公钥，下面 U 和 V 开始协商密钥。

设 U 选择：

$$r_U = 8999$$

相应的 s_U 为：

$$s_U = 5^{8999} \bmod 1\ 091\ 309 = 448\ 786$$

V 选择：

$$r_V = 9001$$

相应的 s_V 为：

$$s_V = 5^{9001} \bmod 1\ 091\ 309 = 924\ 692$$

于是 U 可以计算出密钥：

$$k = 924\ 692^{197\ 503}(659\ 617^{109\ 981} + 200\ 510)^{8999} \bmod 1\ 091\ 309 = 896\ 417$$

V 也能计算出密钥：

$$k = 448\ 786^{197\ 702}(150\ 608^{109\ 981} + 200\ 409)^{9001} \bmod 1\ 091\ 309 = 896\ 417$$

显然，Girault 密钥协商协议的另一个特点是既基于因式分解难题，也基于离散对数问题。

8.4 公钥基础设施

前面介绍了对称密码体制和公钥密码体制，以及利用这些体制完成消息签名、认证、密钥分发和协商等任务。虽然现有的技术和算法都足以完成这些任务，但是无论对消息、身份，还是密钥本身的操作都需要一种规范的框架。本节要讨论的公钥基础设施(Public Key Infrastructure，PKI)正是这样一种框架。

总的来说，PKI 是建立在公钥密码系统的基础之上的，用来给各种需要安全保证的用户提供安全服务的一种解决方案。它具有统一化、标准化的特点，同时还具备良好的互操作性和可扩展性。

用户可以利用 PKI 平台提供的安全服务进行安全通信。PKI 建立在统一的标准和规范基础之上，为网络应用提供实体认证、数据的保密性、数据的完整性和交易的不可否认性等安全服务。

一个完整的 PKI 系统，如图 8.6 所示。它主要由以下几个部分构成。

1. 认证机构(Certificate Authority，CA)

CA 是 PKI 的核心执行机构，是 PKI 的主要组成部分，又称认证中心。

CA 的主要职责如下：

(1) 验证并标识证书申请者的身份。对证书申请者的信用度、申请证书的目的、身份的真实性等问题进行审查，确保证书与身份一致。

(2) 确保用于签名证书的非对称密钥的安全性。为了增加破解的难度，CA 用于数字签名的私钥必须由硬件卡产生，具有足够安全长度的密钥。

(3) 管理证书信息资料。管理证书序号和 CA 标识，确保证书主体标识的唯一性，防止证书主体标识的重复。在证书使用中确定并检查证书的有效性，确保不使用过期或已作废的证书。

(4) 发布和维护作废证书列表(CRL)。如果证书因某种原因需要报废，就必须将其作为"黑名单"发布在证书作废列表中，以供交易时在线查询。对已签发证书的使用进行监视跟踪，以备发生交易争端时提供证据。

由此可见，CA 是保证电子商务、电子政务、网上银行、网上证券等交易的安全性、公正性的第三方权威机构。

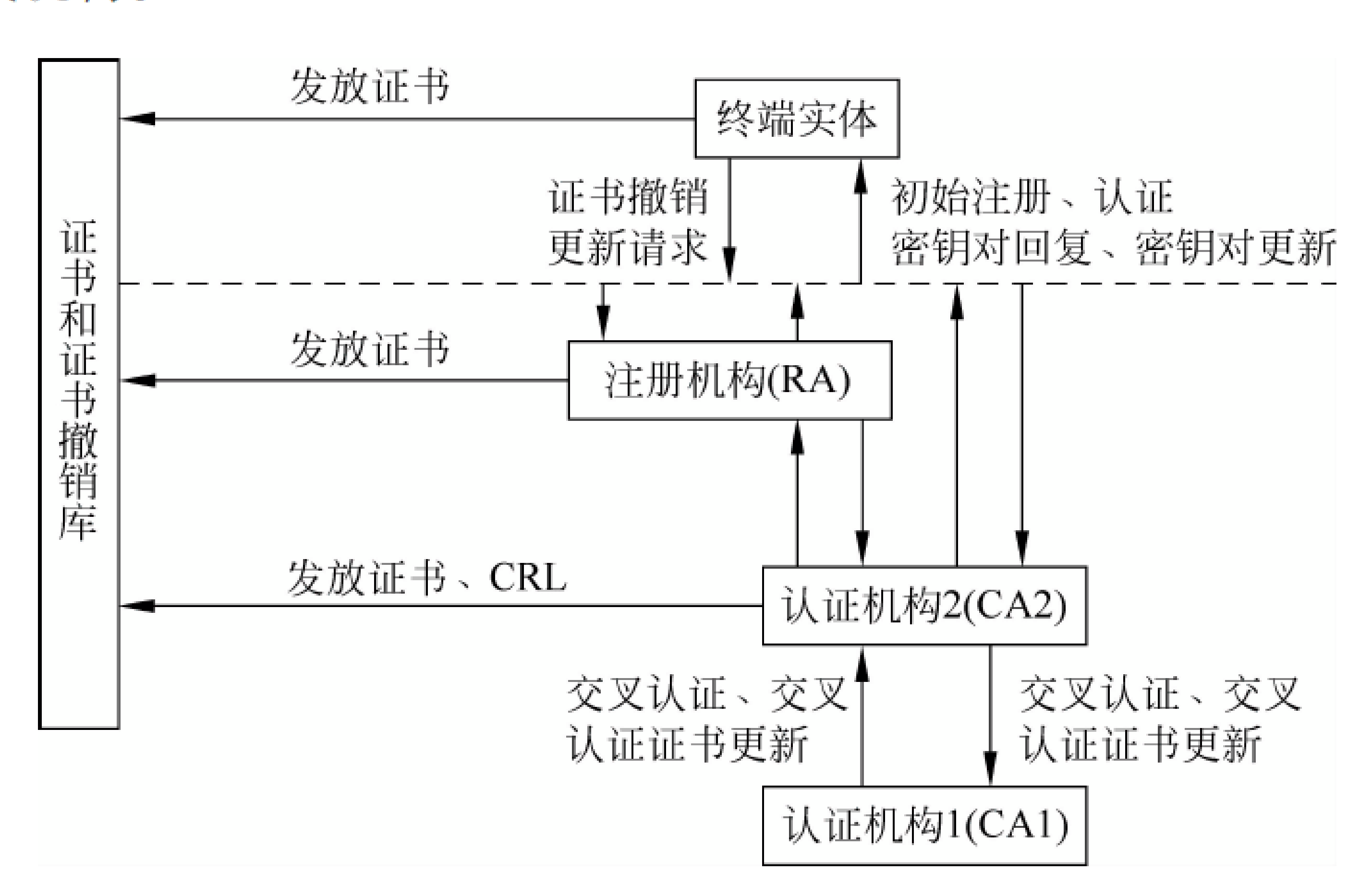


图 8.6　完整的 PKI 系统图

2. 注册机构(Registration Authority，RA)

RA 是可选的管理实体，换句话说就是在 PKI 中可以有 RA 也可以没有 RA，而且有些 PKI 中 RA 独立存在，有些 PKI 中 RA 与 CA 合为一体。

RA 可以看作是终端实体与认证中心 CA 之间的一个接口。接受用户的注册申请，获取并认证用户的身份，完成收集用户信息和确认用户身份。

尽管注册的功能可以直接由 CA 来实现，但当 PKI 域内实体用户数量很大并且在地理上分布很广泛的时候，就有必要建立单独的 RA 来实现注册功能。

3. 证书(和证书撤销列表)库(Cert/CRL Repository)

数字证书是网上实体身份的证明，证书应由具备权威性、可信任性和公正性的第三方机构签发。

证书库是 CA 颁发证书和撤销证书的集中存放地，它像网上的"黄页"一样，是网上的公共信息库，可供公众进行开放式查询。一般来说，查询的目的有两个：一个是想得到与之通信实体的公钥；另一个是要验证通信对方的证书是否在"黑名单"上。证书库一般支持分布

式存放，目的是提高效率，减少总目录查询的压力。

4. 终端实体（End Entity，EE）

终端实体就是PKI中的用户。根据在通信过程中用户的不同类型，终端实体可以分为两类：证书持有者和依赖方（Relying Party）。

为方便客户操作，解决PKI的应用问题，PKI为客户提供客户端软件，以支持数字签名、加密传输数据等功能。此外，客户端软件还负责在认证过程中，查询证书和相关证书的撤销信息以及进行证书路径处理、对特定文档发出时间戳请求等。

下面将重点介绍PKI的两个重要主题：数字证书和信任模型。

8.5 数字证书

数字证书是各终端实体在网上进行信息交流及商务活动的身份证明，在电子商务交易的各个环节，交易各方都需要验证其他参与各方的数字证书的有效性，从而解决相互间的信任问题。

目前，数字证书的格式已形成统一的标准，即X.509标准。

X.509是由国际电信联盟ITU-T制定的数字证书标准。最初的X.509版本公布于1988年，版本3的建议稿于1994年发布，并于1995年获得批准。

在X.509中，数字证书的数据域包括以下内容。

(1) 版本号：如v3。

(2) 序列号：由同一发行者（CA）发放的每个证书的序列号是唯一的。

(3) 签名算法识别符：签署证书所用的算法及相应的参数。

(4) 发行者名称：指建立和签署证书的CA名称。

(5) 有效期：包括证书有效期的起始时间和终止时间。

(6) 主体名称：指证书所属用户的名称，及这一证书用来证明私钥用户所对应的公开密钥。

(7) 主体的公开密钥信息：包括主体的公开密钥、使用这一公开密钥算法的标识符及相应的参数。

(8) 发行者唯一识别符：这一数据项是可选的，当CA名称被重新用于其他实体时，则用这一识别符来唯一标识发行者。

(9) 主体唯一标识符：这一数据项也是可选的，当主体的名称被重新用于其他实体时，则用这一识别符来唯一地识别主体。

(10) 扩展域：其中包括一个或多个扩展的数据项，仅在第3版中使用。

(11) 签名：CA用自己的私钥对上述域的哈希值进行数字签名的结果。

证书由某个可信的证书发放机构（CA）建立，并由CA或用户自己将其放入公共目录中，以供其他用户访问。

目前，符合X.509标准的证书已被广泛应用于网络安全应用程序，其中包括IPSec、SSL、安全电子交易（SET）、S/MIME等。

例 8.5　Microsoft Root Authority 证书。

在 Microsoft 自带的 Internet Explore 软件中，就可以查看到证书，方法如下：

选择“工具”→“Internet 选项”→“内容”选项卡，单击其中的“证书”按钮，于是得图 8.7 所示的证书列表。



图 8.7　Internet Explore 中的证书列表

不妨选择其中一个证书，这是一个名称为“Microsoft Root Authority”的根证书，如图 8.8 所示。



图 8.8　Microsoft Root Authority 证书

可分析该证书的数据域(参见图 8.9)如下。

版本号：

v3，表示版本 3

序列号：

00C1 008B 3C3C 8811 D13E F663 ECDF 40

签名算法识别符：

md5RSA

发行者名称：

CN ＝ Microsoft Root Authority

OU ＝ Microsoft Corporation

OU ＝ Copyright (c) 1997 Microsoft Corp

有效期起始日期：

1997 年 1 月 10 日 15:00:00

有效期终止日期：

2020 年 12 月 31 日 15:00:00

主体名称：

CN ＝ Microsoft Root Authority

OU ＝ Microsoft Corporation

OU ＝ Copyright (c) 1997 Microsoft Corp

主体的公开密钥信息：

如图 8.9 所示，证书的 2048 位公钥如下：

3082 010A 0282 0101 00A9 02BD C170 E63B F24E 1B28 9F97 785E 30EA A2A9 8D25 5FF8 FE95 4CA3 B7FE 9DA2 203E 7C51 A29B A28F 6032 6BD1 4264 79EE AC76 C954 DAF2 EB9C 861C 8F9F 8466 B3C5 6B7A 6223 D61D 3CDE 0F01 92E8 96C4 BF2D 669A 9A68 2699 D03A 2CBF 0CB5 5826 C146 E70A 3E38 962C A928 39A8 EC49 8342 E384 0FBB 9A6C 5561 AC82 7CA1 602D 774C E999 B464 3B9A 501C 3108 2414 9FA9 E791 2B18 E63D 9863 1460 5805 659F 1D37 5287 F7A7 EF94 02C6 1BD3 BF55 45B3 8980 BF3A EC54 944E AEFD A77A 6D74 4EAF 18CC 9609 2821 0057 9060 6937 BB4B 1207 3C56 FF5B FBA4 660A 08A6 D281 5657 EFB6 3B5E 1681 7704 DAF6 BEAE 8095 FEB0 CD7F D6A7 1A72 5C3C CABC F008 A322 30B3 0685 C9B3 2077 1385 DF02 0301 0001



图 8.9 证书的数据域以及公钥

发行者唯一识别符：

KeyID=5b d0 70 ef 69 72 9e 23 51 7e 14 b2 4d 8e ff cb

Certificate Issuer：

CN=Microsoft Root Authority

OU=Microsoft Corporation

OU=Copyright (c) 1997 Microsoft Corp

主体唯一标识符：

Certificate SerialNumber=00 c1 00 8b 3c 3c 88 11 d1 3e f6 63 ec df 40

签名：

SHA1(签名算法)

A434 8915 9A52 0F0D 93D0 32CC AF37 E7FE 20A8 B419

数字证书是一个在 PKI 中用于认证和分发公钥、用户身份信息以及其他信息的基本实体。但是，可能存在某些原因使得证书中的密钥有可能在其使用期内不再有效，如私钥泄露、密钥用途变更、证书主体关系变更等，于是需要一种机制来撤销和更新数字证书的状态，这就是所谓的“证书状态机制”。

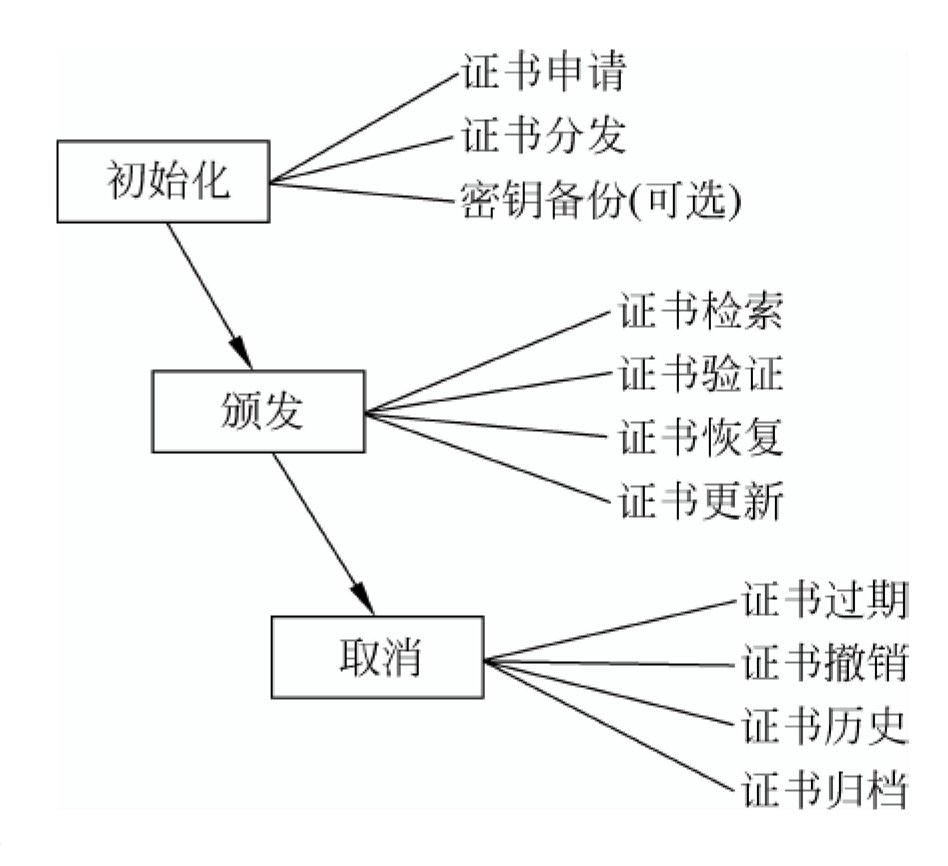


图 8.10　证书的生命期

图 8.10 中显示了数字证书的生命期。

具体过程详细说明如下。

1. 初始化

终端实体被 CA/RA 认证成功的结果就是 CA 为终端实体的公钥签发证书，并把该证书返回给终端实体，并将这个证书分发到公共的存储库中去。

终端实体必须首先得到 CA 的根证书，因为所有的证书验证都将用到它。一般 CA 会在自己的门户网站上发布自己的证书，用户只需登录该网站即可下载 CA 根证书。接下来的过程是证书申请、证书分发和密钥备份。

1) 证书申请

在证书申请阶段，用户必须首先到注册中心 RA 注册证书申请信息，RA 根据相应的策略审核用户申请信息。如果审核通过，RA 向 CA 提出证书签发请求，由 CA 负责证书的签发和发放。

申请证书的第一步是进行用户注册。用户注册一般有两种方式，一种是离线注册方式，即用户持相关有效证件到注册中心 RA 处进行书面登记，按要求填写证书相关表格，类似于到银行的营业厅去申请银行卡。另一种方式是在线的注册方式，许多 CA 都提供一个门户网站，用户可以登录该网站进行证书申请。

在公钥密码体系中，公钥、私钥对的产生是很重要的。用户证书申请的目的就是向 CA 认证自己的身份，认证成功的用户将获得 CA 颁发的公钥证书。密钥对的产生有以下三种方式：

(1) 用户自己产生。用户保存自己的私钥，然后向 CA 传送公钥以及自己的身份证明，请求 CA 对该身份和公钥进行认证，并颁发证书。

(2) 密钥对由第三方产生。

(3) 密钥对由 CA 产生。由于 CA 是一个权威、可信的机构,本身具有极高的安全性,由它来负责密钥的产生是一个很好的选择。

后面两种可以统称为集中密钥产生方式。

集中密钥产生方式可以使密钥备份和恢复之类的服务简单化,不过私钥的安全传输也是一个必须要考虑的问题。

用户自己产生密钥对方式的好处则是用户私钥保存在本地,不需要传输;但是用户也因此无法使用 CA 的密钥备份和恢复服务。如果用户需要 CA 帮助备份自己的加密私钥,也会存在私钥传送的问题。

鉴于以上原因,有些 PKI 系统实现将两种方式结合起来。用于加密和解密的密钥对,使用集中密钥生成方式,CA 根据用户请求决定是否备份该密钥;而用于签名的密钥对则一般由用户自己生成。

如果密钥对由用户自己产生,还必须进行私钥拥有验证(Proof of Possession,POP),验证的目的是确定身份和私钥之间的联系,实现的方法根据证书请求中的密钥类型的不同而不同。

如果密钥是用来签名的,那么用户使用自己的私钥对证书申请请求进行签名,发送给 RA,RA 再用请求中的公钥验证签名即可证明用户确实拥有私钥。由于私钥签名具有不可否认性,因此这种方法可以证明用户确实是私钥的拥有者。

如果密钥是用来加密的,那么由 RA 产生一个随机数据,然后用证书申请请求中的公钥对它加密,用户如果可以解密该加密数据,就可以证明他确实拥有私钥。另一种方法是 CA 返回给用户的公钥证书通过用户提交的公钥进行加密。用户只有用自己的私钥解密才能解密出自己的公钥证书,从而证明自己确实拥有对应的私钥;但这种方法对于证书的发布存在问题,因为证书必须明文发布于公开的存储库中。

2) 证书分发

PKI 提供的安全服务一般都需要证书,如何获取用户证书是证书分发所要解决的关键问题。证书的分发主要体现在 CA 和最终实体的交互以及各个不同最终实体之间的交互。

CA 和终端实体之间的证书分发是指 CA 根证书的分发,CA 在首次初始化时要生成自签名的根证书。客户端必须首先要获得 CA 的根证书,才能安全地使用 CA 提供的安全服务。通常 CA 的根证书是通过自己的门户网站来发布的,用户可以下载 CA 根证书,并安装到浏览器的 CA 证书库中。

用户之间的证书分发分两种情况:一种情况是用户之间为了进行安全通信,需要使用对方的公钥证书;另一种情况是当得到对方的证书后要使用时,需要验证对方证书的状态。

用户之间的证书分发方式包括以下几种:

(1) 私下分发。最简单的分发机制是用户之间的私下分发,用户将自己的证书直接传送自己信任的终端实体,可以采用优盘方式、E-mail 附件方式等。这种方式只适合小范围的用户群体,不适用于当前 Internet 上的安全应用。

(2) 集中分发。最常用的证书分发方式是在一个公共的资料库中公布所有用户证书,使终端实体可以方便、快捷地获取需要与之进行安全通信的对方的公钥证书。这种集中分发服务一般包括以下的服务或服务器:

① 基于 LDAP 的目录服务;

② X.500 目录系统;

③ 在线状态服务器(Online Certificate Status Protocol，OCSP)；

④ Web 服务器；

⑤ 基于 FTP 的服务器。

(3) 协议分发。证书还可以作为某种安全协议交换的一部分来分发。例如,客户端和服务器通过安全套接字(将在后面的章节介绍)进行通信时,服务器端可以配置成要求进行客户端验证,即通过协议要求客户端必须提供自己的证书,此时客户端就可以将自己的证书作为协议交换的一部分来分发自己证书。为了验证客户证书,SSL 还要求服务器可以访问证书链中每一个 CA 证书。此外,IPSec 中的密钥交换协议 IKE 也可以传送证书。

3) 密钥备份

密钥备份是由 CA 或者可信任的第三方提供的一个重要服务。密钥备份是解决密钥丢失并帮助恢复重要数据的唯一方法。

在 PKI 中,终端实体一般至少有两对密钥对,一对是用于签名的签名密钥,另一对是用于加密的加密密钥,每对密钥都有相应的数字证书。签名密钥用于对发送的信息进行数字签名,由于数字签名的不可否认性,签名密钥一般由用户自己产生,密钥备份服务不能备份该密钥。而对于用户的加密密钥,应该进行备份以防止加密数据无法解开造成的损失。

为防止密钥备份管理人员滥用权限造成密钥泄露的危险,备份的密钥应由多个管理人员共同管理,用户的私钥可以被分为若干部分,由不同的密钥管理人员保存,任何一个管理人员都无法恢复用户的私钥,只有他们所持有的分量合在一起的时候才能恢复私钥。这种方法尤其适合对安全性要求特别高的 CA 根密钥的备份,对于普通用户密钥来说,较常用的方法是加密保存。

2. 管理阶段

初始化阶段完成后,签发的密钥和证书需要进行有效的管理,从而进入管理阶段。管理阶段的服务包括证书检索、证书验证、密钥恢复、密钥更新。

1) 证书检索

证书检索的目可能是为了验证签名,也可能是加密数据。

证书的检索主要在目录服务器上进行。使用 LDAP 协议必须在客户端配置相应的支持 LDAP 的客户端程序,许多 CA 还直接利用 FTP 或者 HTTP 实现对证书的检索查询,此时证书以文件形式存储在某服务器中,该服务器的 URI 能提供证书的用户名、组织等信息。

2) 证书验证

在使用证书之前,必须确认该证书是否有效。证书验证是确认证书是否有效的过程,包括确认证书是否由受信任的 CA 签发、证书是否完整而且有效。完整性通过验证证书上的 CA 数字签名来保证,而有效性是指证书未过期而且未被撤销。

证书验证的基本步骤如下：

(1) 证书路径建立。建立一条从待验证证书到可信任 CA 根证书的路径,该路径上的所有证书都应能够被验证者访问。

(2) 证书路径验证。检查路径上的所有证书的数字签名以保证每一个证书都是真实可信的,没有被篡改过。验证从最顶层的证书开始,提取出该证书中的公钥,验证下级证书的数字签名,如果验证通过,则保证了下级证书的完整性和可靠性,通过验证后的下级证书的公钥又可以用来检验其再下一级证书的数字签名,依次类推,直到验证完所有的证书。

（3）证书状态检查。由于证书受到有效期的限制，所以必须保证证书路径上的每一个证书都在有效期之内，并且没有并撤销。验证有效期的方法是提取出证书中的有效期，然后与当时时间进行比较，如果当时时间在有效期的范围之内，则说明证书有效。

有效期的验证也必须从根证书一直到被验证的证书，即只有上面各级证书在有效期内，被验证证书在有效期内才有意义。

检查证书是否被撤销有两种方法：一种方法是下载 CA 中心签发的最新 CRL，检验证书路径上所有证书的序列号是否在证书撤销列表中，只要出现了任何一个，那么被验证证书就是无效的；另一种方法是使用在线证书状态查询 OCSP，发送证书状态查询请求给 OCSP 服务器，OCSP 服务器会返回所请求证书的状态“有效”、“已撤销”或者“不确定”。

3）密钥恢复

密钥恢复是基于密钥备份来实现的，没有备份的密钥是无法恢复的。

当终端实体的加密密钥丢失后，可以请求恢复密钥，从而避免重要数据的丢失。密钥恢复应自动完成，最大可能地减少用户的参与。

4）密钥更新

密钥更新是在证书中密钥过期之前重新签发新的密钥和证书的过程，也称为证书更新。最理想的密钥更新是在密钥达到生命期的 3/4 时自动并透明完成的，从而使得新旧密钥能够平滑地进行切换，防止服务的中断。

密钥更新的原因除了密钥快到生命周期外，还包括证书中的一些属性发生变化、私钥泄露等。

密钥更新包括最终实体密钥更新和密钥对更新两种情况。

（1）最终实体密钥更新。虽然 CA 可以为用户自动实现自动密钥更新，即在用户使用证书的过程中，CA 自动到目录服务器中检查证书的有效期，对于快到期的证书完成自动更新。但在某些情况下，如前面提到的证书中的一些属性改变，用户会向 CA/RA 提出证书更新的请求。证书更新请求基本上同证书申请的过程一样，唯一不同的是证书更新需要提供更新证书的序列号。

（2）密钥对更新。PKI 系统的信任关系是建立在 CA 的密钥对上的，当 CA 需要更新密钥对时，整个 PKI 系统都将受到影响。在 CA 发布新的根证书时，需要把所有用户对原密钥的信任关系转移到新密钥上来。CA 的用户群非常巨大，所有证书的验证、管理都需要用到 CA 的根证书，如何实现 CA 根证书的切换，是 CA 密钥更新中的关键问题。

RFC2510 给出了 CA 密钥更新方法，具体步骤如下：

① 产生新的密钥对；

② 使用新私钥为旧公钥签发证书，称为“新私钥旧公钥”证书；

③ 使用旧私钥为新公钥签发证书，称为“旧私钥旧公钥”证书；

④ 使用新私钥为新公钥签发证书是新的自签名证书，称为新 CA 根证书，CA 密钥更新完成后，所有的信任关系将转移到此证书上。

将②、③、④产生的新证书通过目录或者其他方式发布，保证 CA 所有用户能方便地访问。

上述几个步骤完成后，CA 旧私钥将不再需要，而旧 CA 根证书还将继续使用一段时间，直到所有的 CA 用户都已经获得了新 CA 根证书。新 CA 根证书的有效期的开始时间是新密钥对的产生时间，结束时间是下次 CA 密钥对更新的时间。

在 CA 证书更换期间的过度期内，由于“新私钥旧公钥”证书、“旧私钥旧公钥”证书和新

CA 根证书同时存在，无论通信双方的证书新旧与否都能正确使用。

3. 取消阶段

取消阶段是密钥/证书生命周期的结束阶段，它是一个取消的过程。该阶段包括证书过期、证书撤销、密钥历史和密钥归档 4 个部分。

1）证书过期

证书过期发生在证书有效期结束的时候。

每个证书都有一个固定的生命期，期满的证书将不再有效。如果某个证书期满时，继续使用不会造成安全威胁，可以通过延长该密钥对的使用有效期，具体做法是重新签发一个新证书，该证书中包含的是过期证书中的公钥，新证书和过期证书的唯一不同点在于新证书具有新的有效期。

2）证书撤销

数字证书的功能如同现实社会中的身份证。如果身份证丢失，需要到相关部门去挂失。

数字证书也是如此，如果在证书有效期内需要提前中止该证书的使用时，就必须存在一种机制来撤销该证书。

CA 在收到证书撤销请求后，立刻撤销终端实体的证书并及时发布该撤销信息，警告所有的证书使用者，该证书不再代表一个原终端实体的身份。

证书撤销的实现方法有多种：一种方法是采用证书撤销列表公布证书撤销的信息；另一种方法采用在线查询机制，实时公布被撤销的证书。

证书撤销列表 CRL 实质上是一个由签发证书的、以所定期签发的带 CA 签名和时戳的数据结构，它包含被撤销的证书的序列号、撤销日期以及撤销原因。

CRL 的公布间隔可能是一天也可能是一周。在 CRL 中包含有下一个 CRL 的发布时间。

CRL 的发布方式并不统一，它也是由各自的 CPS 规定的。总的来说，CRL 的发布方式包括以下 3 种：

（1）完全 CRL 方式。每次发放的 CRL 包括所有已被撤销的证书，其文件大小随时间的增加而增长。

（2）增量 CRL 方式。每次发放的 CRL 不包括已被撤销的证书，只包括上一个 CRL 的增量。这种方式避免了每撤销一个证书就必须产生一个庞大的 CRL。

（3）分布式方式。为了控制 CRL 的大小，将 CA 颁发的证书分成多个子集，每个子集中的证书撤销由不同的 CRL 来发布。该方式通过每个证书中的扩展项字段——CRL 分布点来实现，分布点为终端实体确定了可以在什么地方获得证书的状态信息。

CRL 可以采用多种方式分发，如目录服务、E-mail 和 Web 形式，但一般同证书的分发方式保持一致。

OCSP 是另一种方式。即终端实体通过 OCSP 客户端发送一个证书的状态查询给 OCSP 服务器，OCSP 服务器返回该证书的状态。

OCSP 一般用于网上银行、网上证券、电子政务中的某些关键应用。除了返回证书状态信息之外，基于 OCSP 的在线验证往往还可以返回关于证书的其他信息。

3）密钥历史

由于密钥更新、证书过期等原因，每个终端实体都可能拥有多个旧的公钥证书和当前的证书。这一系列证书构成了用户的证书历史档案，称为用户密钥历史。保存密钥历史是恢复以前加密的敏感数据的可行方案。

通常使用一个另外的独立系统来提供密钥历史服务，但需要向其他系统提供接口，接口的数目将根据实际的 PKI 要求而定。

如果密钥对是由用户自己产生，则需要提供同用户的接口；如果密钥对由 CA 集中产生，则需要提供同 CA 的连接接口。

4）密钥归档

密钥归档不同于密钥历史，密钥历史是面向数据机密性服务，它通常只关心终端实体用于加解密的密钥的存储，帮助终端实体恢复已经过期的密钥。

密钥归档一般是由第三方提供的服务，它除了涉及用户的密钥的保存外，还可能涉及其他同用户相关的秘密资料的存储。如果将密钥归档服务和安全时间戳、安全公证服务结合在一起，还能提供审计和帮助解决争议的服务。

例 8.6　Windows 环境下的证书服务安装与配置。

证书的安装如图 8.11 和图 8.12 所示。

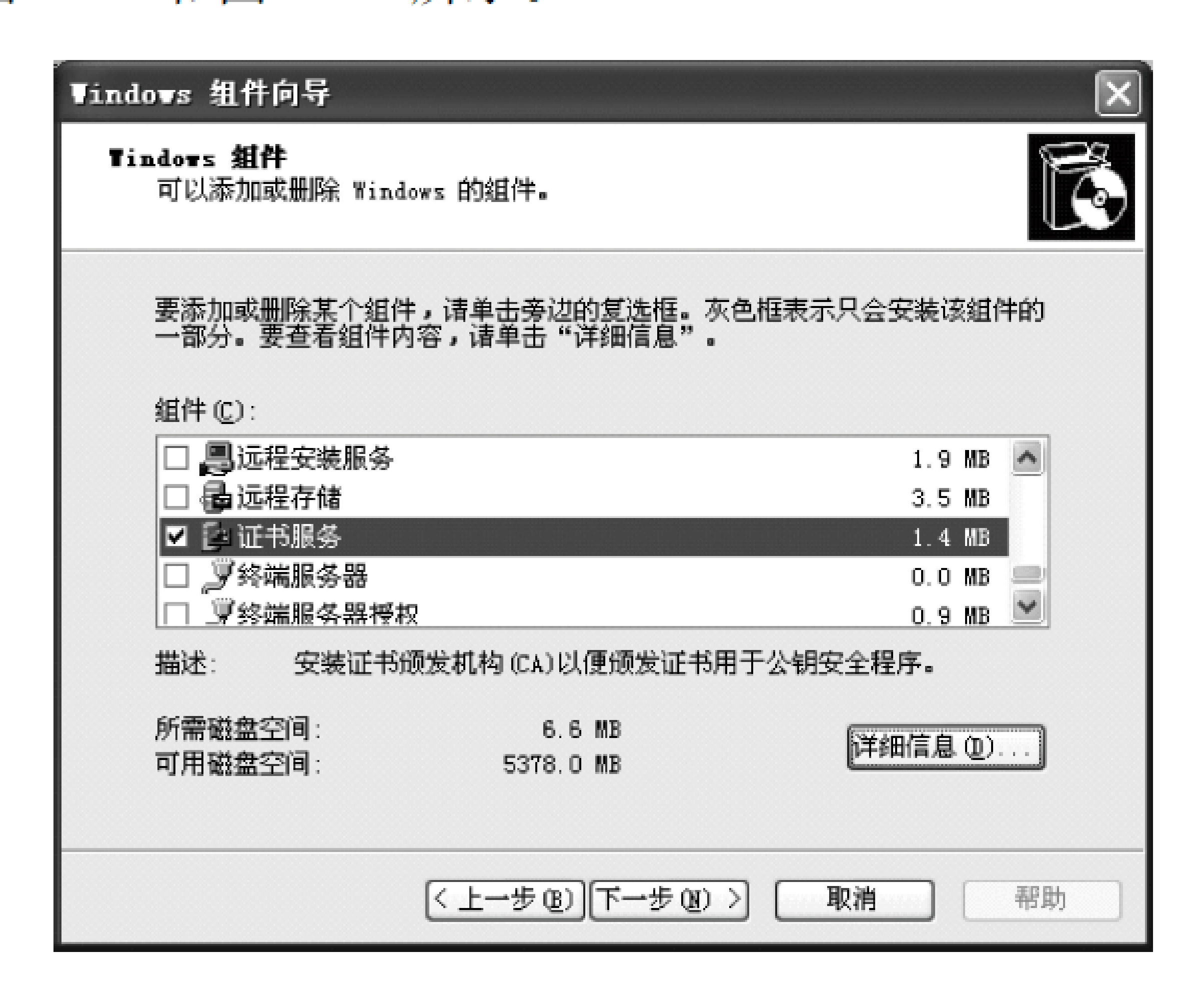


图 8.11　添加 Windows 组件



图 8.12　证书数据库设置

安装成功后，用户可以通过浏览器申请证书，如图 8.13 所示。

CA 证书恢复如图 8.14 和图 8.15 所示。

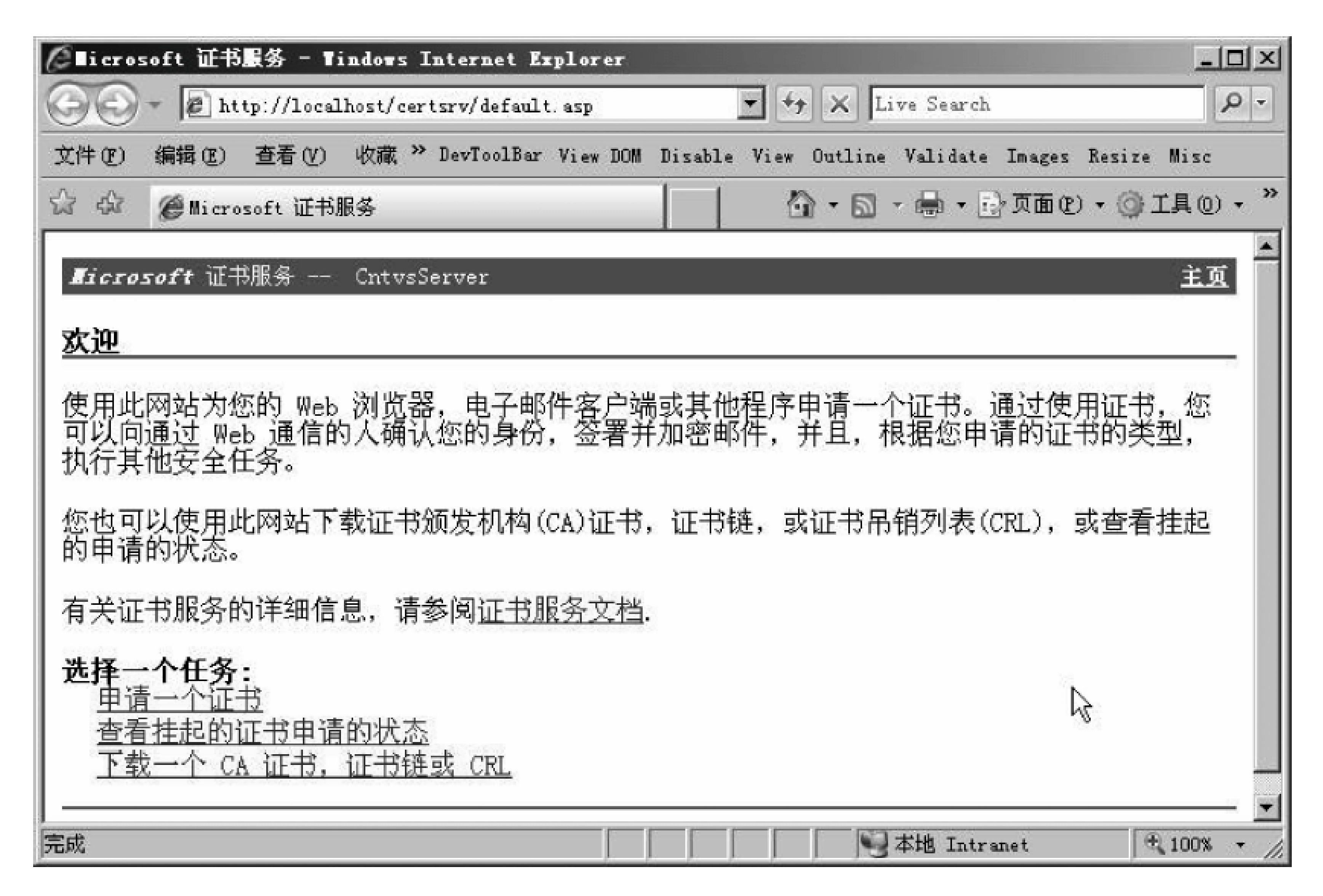


图 8.13　用户申请证书

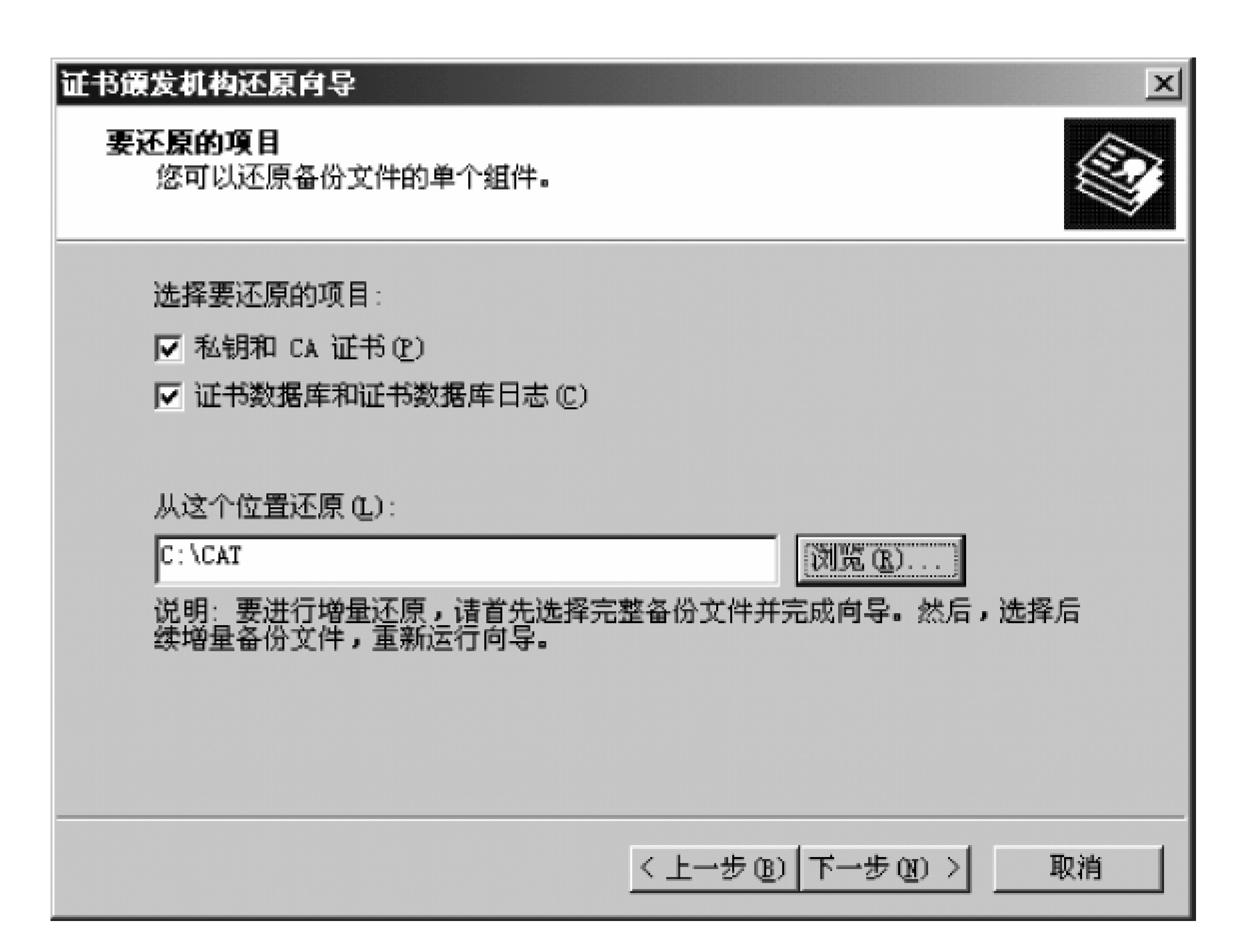


图 8.14　证书恢复过程中要还原的项目

证书颁发机构还原向导

提供密码

要加密和解密消息，需要公钥和私钥。您必须提供私钥的密码。

需要这个密码来访问私钥和 CA 证书文件。

密码(P)：

******

为了维护私钥安全，请不要共享您的密码。

< 上一步(B)　下一步(N) >　取消

图 8.15　用户提供密码

8.6 信任模型

近年来，PKI 的建设已按区域和行业全面铺开。区域型的 CA，如上海 CA 中心（SHECA）、北京 CA（BJCA）、海南电子商务认证中心（HNECA）等。行业型的 CA 有国内 13 家商业银行联合建设中国金融认证中心（CFCA）、中国电信组建的 CTCA、由国家外经贸部建立的中国国际电子商务中心（CIECC）等。

事实上，每个 CA 只可能覆盖一定的范围，不同系统往往有各自不同的 CA，他们颁发的证书只在本系统范围内有效。

虽然不同的系统具有自己的 PKI 体系结构，但这些不同的 PKI 体系在实际中又是相互联系的，信任模型主要研究如何解决单个 PKI 体系中的信任问题以及各多个独立 PKI 体系间的交叉信任问题。

1. 单 CA 信任模型

单 CA 信任模型，如图 8.16 所示。

单 CA 信任模型是最基本的信任模型，也是在企业环境中最典型的一种模型。在这种模型中，整个 PKI 体系只有一个 CA，它为 PKI 中的所有终端实体签发和管理证书。每个证书路径都起始于该 CA 的公钥，该 CA 的公钥成为 PKI 体系中唯一的用户信任锚。

信任锚是指在信任模型中，当有一个足够可信的身份签发者证明该实体的身份时，才能做出信任该身份的决定。这个可信的身份签发者就是信任锚。

单 CA 信任模型容易实现，易于管理，但是不太适合终端用户群体很大的情况。

2. 分级信任模型

分级信任模型，如图 8.17 所示。

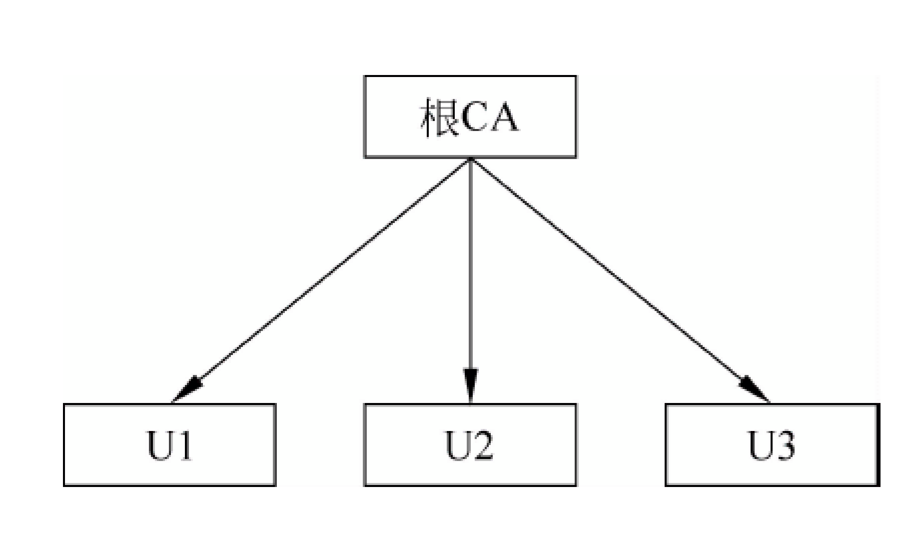


图 8.16 单 CA 信任模型

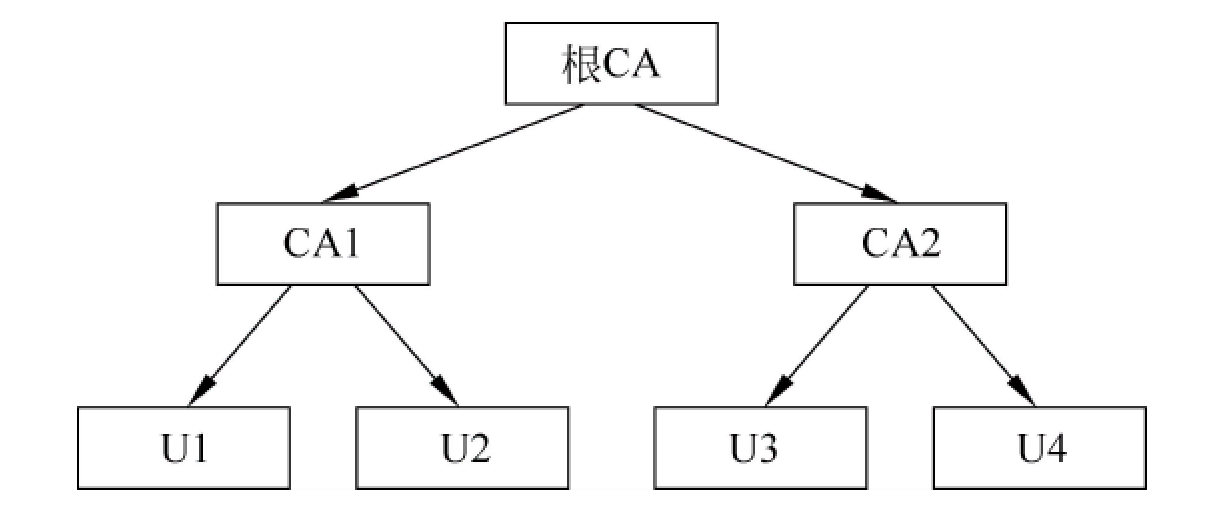


图 8.17 分级信任模型

分级信任模型也称为层次信任模型，可以描绘为一棵倒转的树，根在顶上，树枝向下伸展。在这棵倒转的树上，根代表根 CA，作为信任锚，所有实体都信任它。

根 CA 往往不直接为终端用户颁发证书而仅为子 CA 颁发证书。在根 CA 的下面是多层的子 CA，子 CA 是所在实体集合的根，非 CA 的树叶是指终端实体。两个不同的终端用户进行交互时，双方都提供自己的证书和数字签名，通过根 CA 来对证书进行有效性和真实性的认证。信任关系是单向的，上级 CA 可以认证下级 CA，而下级 CA 不能认证上级 CA。

分级信任模型有以下一些优点：

（1）增加新的 CA 或终端实体比较容易；

（2）证书路径由于其单向性，生成从终端用户证书到信任锚的路径简单明确；

（3）证书路径相对较短，最长的路径等于树的深度加 1。

分级信任模型的缺点是单个 CA 的失败会影响整个 PKI 体系。与顶层 CA 的距离越短，造成的混乱越大。由于所有的信任都集中在根 CA，一旦根 CA 出现故障，将带来毁灭性的灾难。

3. 网状信任模型

网状信任模型也称为分布式信任模型，在这种模型中 CA 间存在着交叉认证。与分级信任模型不同，网状信任模型把信任分散到两个或更多个 CA 上。

网状信任模型的优点如下：

（1）因为存在多个信任锚，单个 CA 的失效不会影响到整个 PKI；

（2）增加新的 CA 更加容易。

不过，在网状信任模型中，信任路径的确定比较困难。从终端用户证书到信任锚建立证书的路径存在多种选择，有些选择可以形成正确路径，而其他选择则会导致失败。更有甚者，在网状模型的 PKI 中甚至可能会形成一条证书环路。

4. 桥 CA 信任模型

桥 CA 信任模型，如图 8.18 所示。

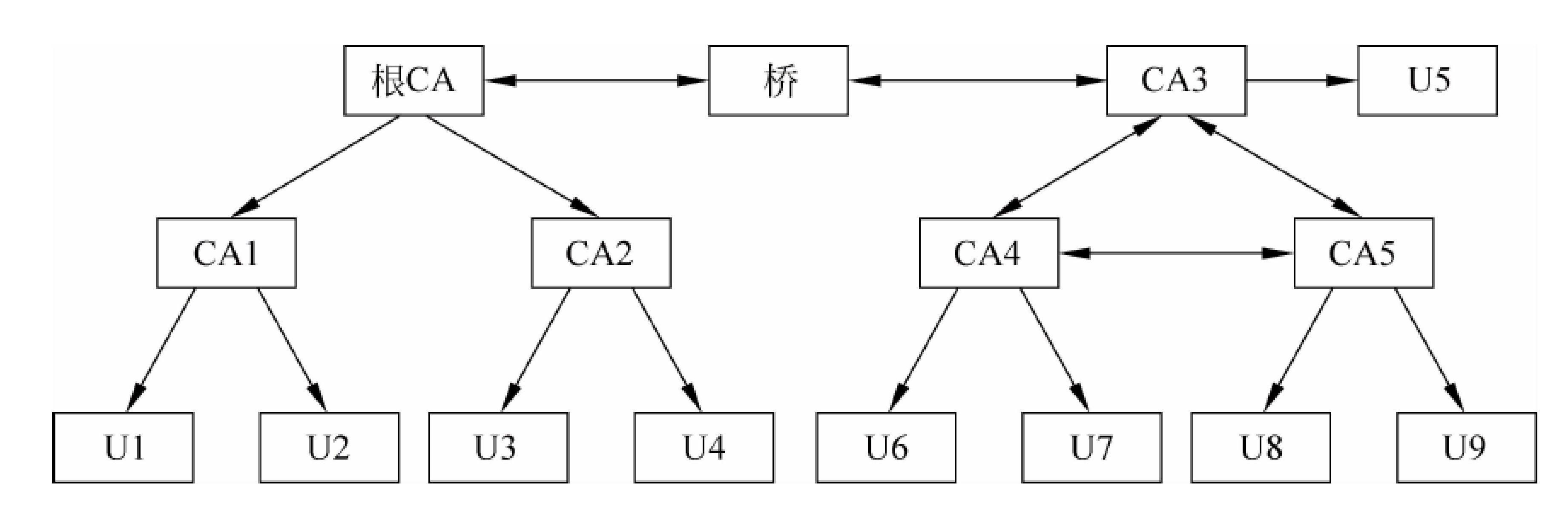


图 8.18　桥 CA 信任模型

桥 CA 信任模型也称为中心辐射式信任模型，它结合了分级模型和网状模型的特点，可以较好地连接不同的 PKI。

不同于网状模型中的 CA，桥 CA 与不同的信任域建立对等的信任关系，允许用户保持原有的信任锚，这些关系被连接起来形成信任桥。

桥 CA 既不直接向用户颁发证书，也不像根 CA 一样能成为一个信任锚。它只是一个单独的 CA，与不同的信任域建立对等的信任关系，允许终端实体保留自己的原始信任锚。

桥 CA 信任模型继承了分级模型和网状模型的优点，不过证书路径的有效发现和确认仍然不很理想。因为基于桥 CA 模型的 PKI 系统可能仍然包括部分的网状模型，这就要求终端实体能开发和确认复杂的证书路径。

5. Web 信任模型

Web 信任模型，如图 8.19 所示。

这种模型一般建立在浏览器，浏览器厂商在浏览器中内置了多个根 CA，每个根 CA 相

互间是平行的,浏览器用户信任这些根 CA 并把它们作为自己的信任锚。

这种模型表面上与分布式信任模型颇为相似,实际上则更接近分级模型。各个嵌入的根 CA 并不被浏览器厂商显式认证,但浏览器厂商隐含认证了这些根 CA。这种依赖关系表明,浏览器厂商就成了事实上的隐含的根 CA。

Web 信任模型的优点是方便简单,操作性强,对终端用户实体的要求较低。用户只需简单地信任嵌入的各个根 CA 即可。

Web 信任模型的缺点是安全性不好,终端实体难知道某个浏览器中嵌入了哪些根 CA。

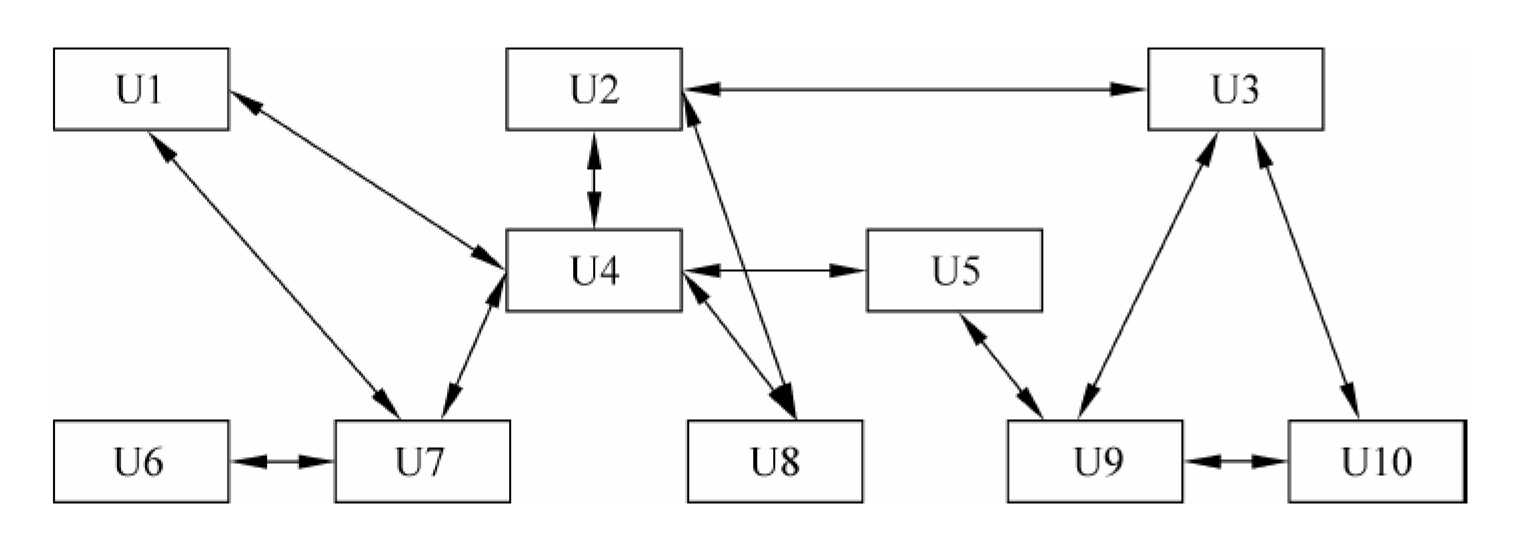


图 8.19 Web 信任模型

6. 以用户为中心的信任模型

在这种信任模型中,终端实体自行决定信赖哪个证书。没有可信的第三方作为 CA,终端实体就是自己的根 CA。

用户信任模型的优点是安全性掌握在自己的手中,用户的自主性强。

用户信任模型的缺点是适用范围小,将发放和管理证书的任务交给普通用户也不现实。

8.7 习 题

1. 在 A、B、C、D 之间实施 Blom 密钥分发方案,具体参数为 $p=419$,$r_A=23$,$r_B=17$,$r_C=43$,$r_D=41$,TA 选择随机数:$a=108$,$b=121$,$c=138$,$d=73$,求 A 与 B、C 与 D、A 与 D、B 与 C 之间通信的共享密钥。

2. 简述 PKI 的工作原理。

3. 简述数字证书的作用。

4. 当通信双方需要通信时,比较成熟的做法是在现有条件下,产生一个临时的通信密钥,这个密钥也称为________。

5. Girault 密钥协商协议包括两个过程,第一个过程是________,第二个过程________。

6. PKI 是建立在________密码系统的基础之上。

7. 如果证书因某种原因需要报废,就必须将其作为发布在________中,以供交易时在线查询。

8. ________可以看作是终端实体与认证中心 CA 之间的一个接口。

9. ________是 CA 颁发证书和撤销证书的集中存放地,它像网上的"黄页"一样,是网上的公共信息库,可供公众进行开放式查询。

10. ________是各终端实体在网上进行信息交流及商务活动的身份证明。

11. 申请证书的第一步是用户注册，用户注册一般有________和________两种方式。

12. 如果密钥对由用户自己产生，还必须进行________验证，验证的目的是确定身份和私钥之间的联系。

13. 有效期的验证也必须从________一直到________，只有上面各级证书在有效期内，被验证证书在有效期内才有意义。

14. CRL 有________、________和________3 种发布方式。

15. 在 CA 证书更换期间的过度期内，由于________证书、________证书以及________根证书同时存在，无论通信双方的证书新旧与否都能正确使用。

16. 分级信任模型也称为________，可以描绘为一棵倒转的树，根在顶上，树枝向下伸展。

第 9 章　计算机安全专题

数字化、网络化是 21 世纪社会生活的一大特点，如何应对日趋严峻的计算机安全威胁备受关注。本章将通过专题的形式，重点介绍口令保护、电子邮件安全、虚拟专用网等几个方面的安全技术。

9.1　操作系统口令保护

计算机离不开操作系统，操作系统的安全是计算机安全的核心问题之一。操作系统最基本的保护是登录口令的保护，本节将针对 UNIX 和 Windows 两种操作系统分别介绍口令保护。

9.1.1　UNIX 系统口令保护

/etc/passwd 文件是 UNIX 口令保护的关键文件之一，当用户登录 UNIX 时往往需要通过该文件来验证口令是否正确。

/etc/passwd 文件是一个文本文件，仅对 root 可写，每行的一般格式为：

```
Username: Passwd : UID : GID : UserInfo : Home : Shell
```

其中，每行的头两项是登录名和加密后的 UNIX 口令，UID 和 GID 是用户的 ID 号和用户所在组的 ID 号，UserInfo 是系统管理员写入的有关该用户的信息，Home 是用户的主目录，Shell 是用户登录后将执行的 shell(若为空格则默认为/bin/sh)。

以下是 Passwd 文件的一个实例：

```
steve:xyDfccTrt180x,M.y8:0:0:admin:/:/bin/sh
restrict:pomJk109Jky41,.1:0:0:admin:/:/bin/sh
pat:xmotTVoyumjls:0:0:admin:/:/bin/sh
```

在口令文件中可以看到，有些加密后的口令有逗号，逗号后有几个字符和一个冒号。逗号后的这个字段与口令维护有关，UNIX 一般要求用户定期地改变他们的口令。

例如，用户 steve 的口令逗号后有 4 个字符“M.y8”，第一个字符表示口令有效期的最大周数，第二个字符决定了用户再次修改口令之前，原口令应使用的最小周数，其余字符表示口令最新修改时间。

Passwd 文件中，时间的编码规则比较特殊：

```
.=0   /=1   0-9=2-11   A-Z=12-37   a-z=38-63
```

系统管理员将前两个字符放进/etc/passwd 文件，以要求用户定期的修改口令；另外两个字符当用户修改口令时，由 passwd 命令填入。

关于前两个字符，还有以下两种特殊情况：

(1) 如果最大周数(第一个字符)小于最小周数(第二个字符)，则不允许用户修改口令，仅超级用户可以修改用户的口令；

(2) 第一个字符和第二个字符都是"."，这时要求用户下次登录时修改口令，修改口令后，passwd 命令会将"."删除，此后不再要求用户修改口令。

用户 restrict 的口令逗号后有两个字符".1"，表明不允许用户修改口令，仅超级用户可以修改用户的口令。

如果对用户的口令维护没有要求，那么可以不加逗号及逗号后的参数，如用户 pat 的记录后面就没有带逗号及参数。

近年来，越来越多的 UNIX 系统口令文件进行了 Shadow 变换，即把/etc/passwd 文件中的口令域分离出来，单独存在/etc/shadow 文件中，并加强对 shadow 文件的保护，以增强口令安全。如果采用 shadow 文件来加强口令安全，那么在/etc/passwd 文件中的口令域中将是一个"x"。

UNIX 使用 DES 加密算法来产生口令验证串，如图 9.1 所示。

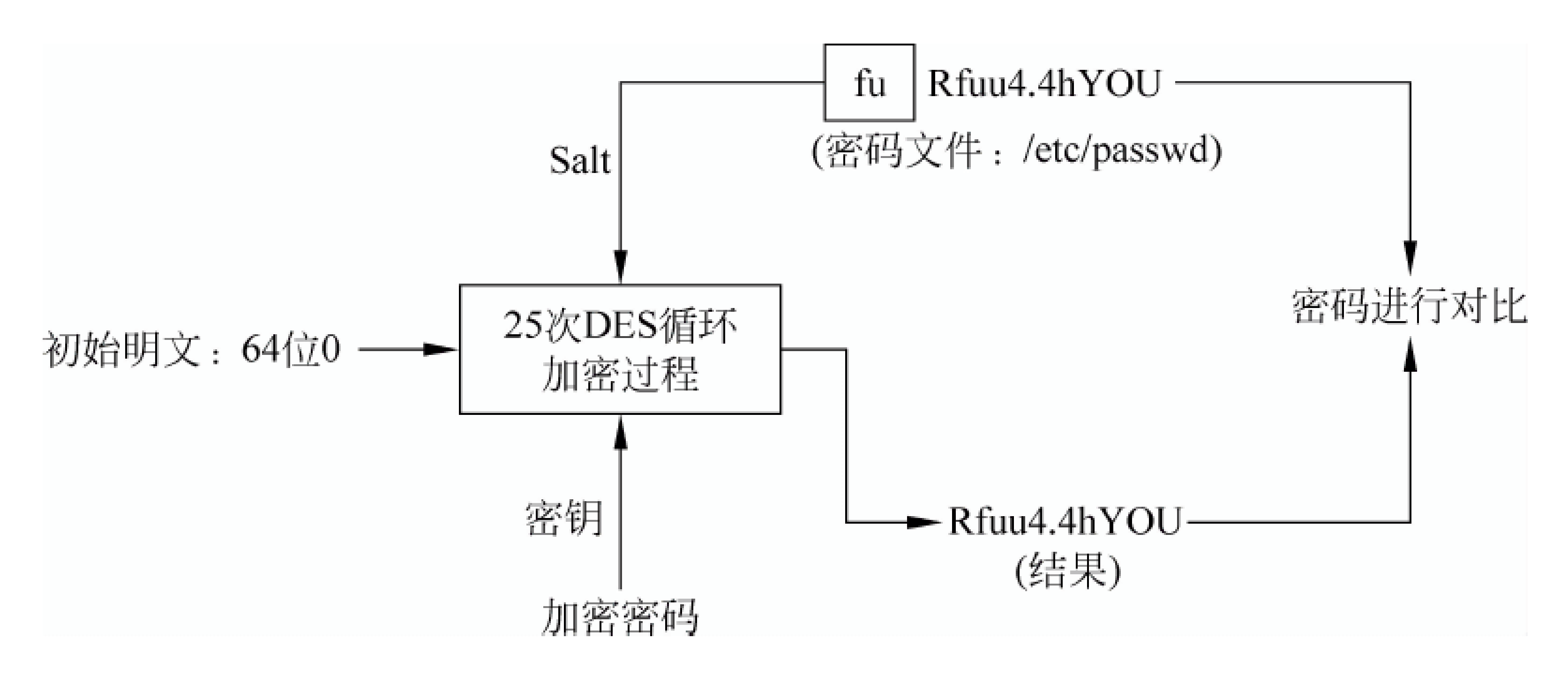


图 9.1　UNIX 口令验证

具体操作如下：

(1) 系统为每个用户准备了一份 Salt 值，Salt 值是 12 位，设 Salt 值 $s=(s_0, s_1, \cdots, s_{11})$。

(2) 初始明文是 64 位的 0。

(3) DES 加密过程需要进行 25 次，这里对 DES 加密进行了小的改动：

在 DES 的子密钥生成运算 $(x_0, x_1, \cdots, x_{31}) \rightarrow (y_0, y_1, \cdots, y_{47})$ 中，如果 $s_j=1$，则 y_j 与 y_{j+24} 互换位置；如果 $s_j=0$，则位置不变。

(4) 最后的 64 位结果分成两部分，一部分由 10 个 6 位构成，另一部分由剩下的 4 位构成，两部分加起来是 11 个 ASCII 字符。

(5) 将 12 位的 Salt 值可转换成两个 ASCII 字符。将 Salt 和 11 个 ASCII 字符连接形成 13 个 ASCII 字符，这 13 个 ASCII 字符存放在口令域中。

9.1.2　Windows 系统口令保护

与 UNIX 不同，Windows 系统的口令保护采用 LM-Hash 或 NTLM-Hash 算法。

Windows 系统的口令文件 SAM 存放在%systemroot%\system32\config 下。由于不是标准的文本文件，要从 SAM 中得到 LM-Hash 或 NTLM-Hash 值，需要借助工具。

下面先介绍 LM-Hash 的运算步骤。

(1) 假设明文口令是"RAYBUCT",首先全部转换成大写 RAYBUCT,再把 RAYBUCT 转换成十六进制形式:

```
"RAYBUCT" → 0x 52415942554354 00000000000000
```

在明文口令不足 14 字节的情况下,后面添加 0x00 补足。

(2) 将上述 14 字节切割成两组 7 字节的数据,每 7 位末尾加一位“0”,把两组 7 个字节变成两组 8 字节。

例如,52415942554354:

```
01010010  01000001  01011001  01000010
01010101  01000011  01010100
```

变成:

```
0101001(0)  0010000(0)  0101011(0)  0010100(0)
0010010(0)  1010101(0)  0000110(0)  1010100(0)
```

即:

```
5220562824AA0CA8
```

00000000000000 则变成:

```
0000000000000000
```

(3) 两组 8 字节数据将作为 DES 的密钥对给定字符串:

```
KGS!@#$%
```

进行标准 DES 加密。

5220562824AA0CA8 对 4B47532140232425 进行 DES 加密的结果是:

```
91A3D48AC422DE92
```

0000000000000000 对 4B47532140232425 进行 DES 加密的结果是:

```
AAD3B435B51404EE
```

(4) 将加密后的这两组数据简单拼接,就得到了最后的 LM-Hash 值:

```
91A3D48AC422DE92 AAD3B435B51404EE
```

为加强口令保护,微软从 Windows NT 3.1 开始使用新的 NTLM-Hash 算法。不过在目前很多版本的 Windows 系统中,LM-Hash 和 NTLM-Hash 依然并存。

NTLM-Hash 的特点是引入了 Unicode,并且口令对大小写敏感。其步骤如下:

(1) 假设明文口令是" RAYBUCT ",先将其转换成 Unicode 字符串:

```
5200410059004200550043005400
```

从 ASCII 字符串转换成 Unicode 串时,使用小字节(little-endian)序。0x80 之前的 ASCII 码转换成 Unicode 码,在原有每个字节之后添加 0x00 即可。

（2）对所获取的 Unicode 串进行标准 MD4 函数处理，MD4 固定产生 128 位的 Hash 值，相当于 16 个字节。

5200410059004200550043005400 的 MD4 值是：

```
E11F3C0699D26C6A68ADC0BB2D250FEC
```

也就是最后的 NTLM-Hash 值。

图 9.2 是借助工具软件 SAMInside 显示的 SAM 文件，在每条记录中同时记录了口令的 LM-Hash 值和 NTLM-Hash 值。

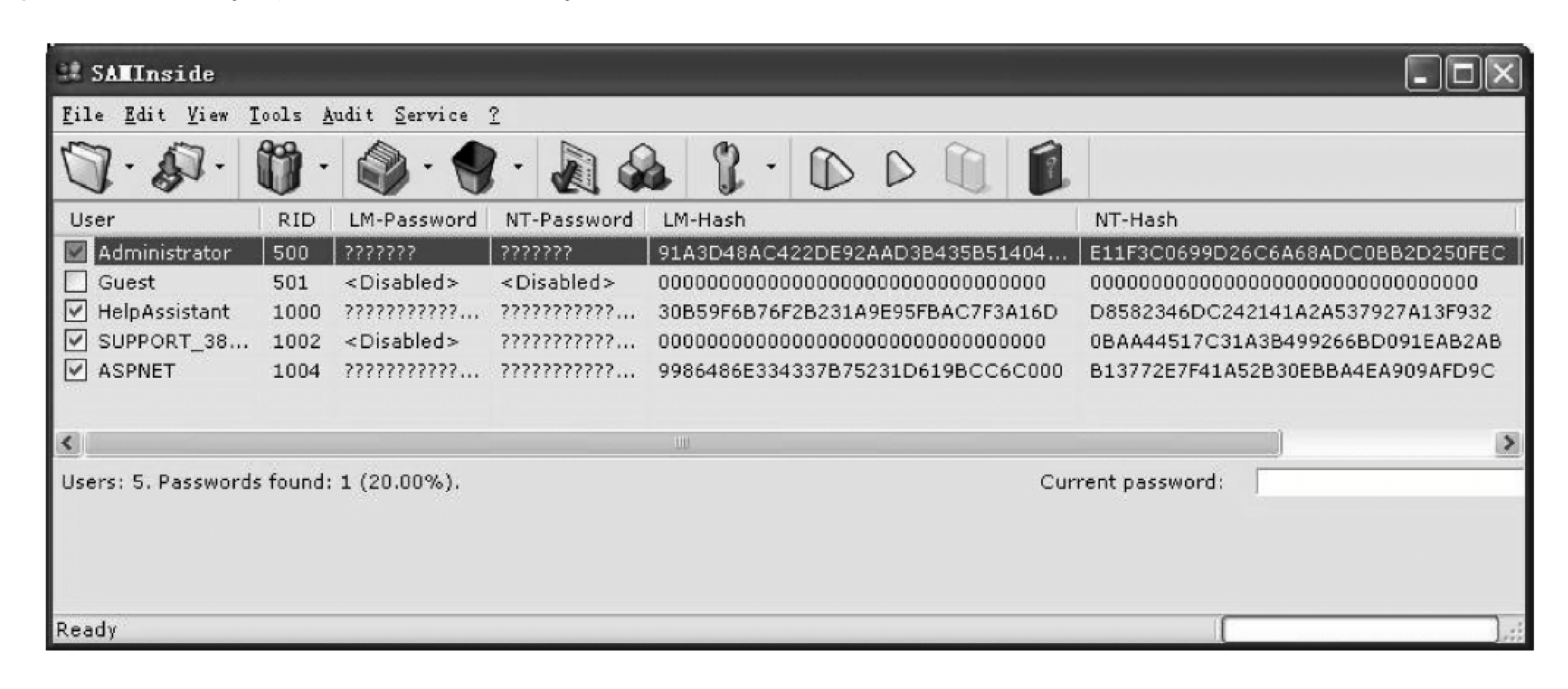


图 9.2　Windows 系统下 SAM 文件的 LM-Hash 值和 NTLM-Hash 值

9.2　PGP 与电子邮件安全

电子邮件对于人们的工作和日常生活来说至关重要，如何保障电子邮件的安全是一个重要的安全主题，使用 PGP 是目前最流行的电子邮件保护手段。

PGP 的创始人是美国的 Phil Zimmermann，他的创造性在于把 RSA 公钥体制的方便与传统加密体系的高效结合起来，并且在数字签名和密钥认证管理机制上有完善的设计，因此 PGP 已经成为一种事实上的标准。

虽然 PGP 总是容易与电子邮件联系在一起，但实际上 PGP 保护的范围不仅仅是电子邮件，PGP 还能够实现以下安全服务。

1. 认证

PGP 使用 RSA 的数字签名和 SHA-1 数字摘要算法实现发送者的消息验证。认证的具体步骤如下：

（1）发送者创建报文；

（2）发送者使用 SHA-1 生成报文的 160 位 Hash 值（数字摘要）；

（3）发送者使用自己的私钥，通过 RSA 算法对数字摘要进行签名，并将其插入到报文的前面；

（4）接收者使用发送者的公钥，采用 RSA 签名的验证算法进行解密，并恢复出数字摘要；

（5）接收者为报文生成新的数字摘要，并与被解密的数字摘要比对，如果两者一致，则报文作为已认证的报文而被接收。

PGP 认证的过程，如图 9.3 所示。图中，M 表示步骤（1）中创建的报文消息，H 表示步骤（2）中的 Hash 函数，EP 表示步骤（3）中的加密（签名）处理，‖ 表示将签名插入到消息前面，Z 表示可选的压缩处理，Z^{-1}表示可选的解压缩处理，DP 表示步骤（4）中的解密（认证）处理。此外，KR 表示发送方的私钥，KU 表示发送方的公钥。

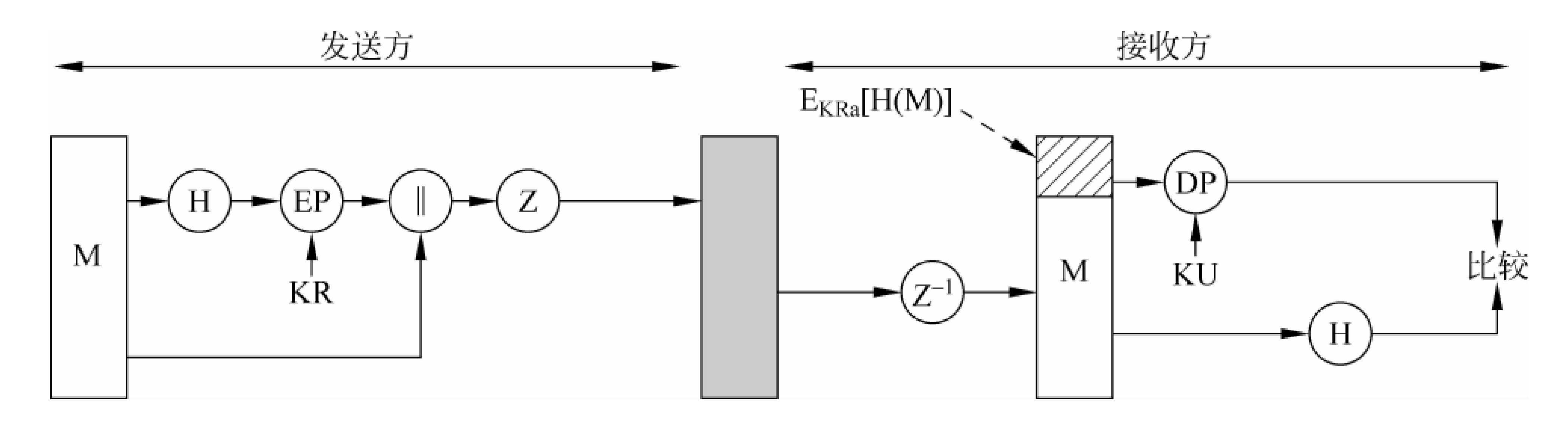


图 9.3 PGP 认证过程

2. 机密性

PGP 采用传统的对称密钥算法进行数据加密，为了安全每个会话密钥只使用一次。会话密钥随机产生，且为了保护会话密钥，发送者使用接收者的公钥对它进行加密。

PGP 既可以对要传递的消息进行加密，也可以对要存储的本地文件实施加密，提供机密性服务。具体支持的对称密钥算法包括 CAST-128、IDES 和 3DES，并使用 64 位密码反馈模式（CFB）。

具体步骤描述如下：

（1）发送者生成报文和用作该报文会话密钥的 128 位随机数。

（2）默认情况下，发送者采用 CAST-128 加密算法，使用会话密钥对报文进行加密。

（3）发送者采用 RSA 算法，使用接收者的公开密钥对会话密钥进行加密，并附加到报文前面。

（4）接收者采用 RSA 算法，使用自己的私钥解密，以恢复会话密钥。

（5）接收者使用会话密钥解密报文。

PGP 机密性服务过程，如图 9.4 所示。图中，EC 表示步骤（2）中的加密过程，DC 表示步骤（5）中的解密过程，Ks 表示会话密钥。

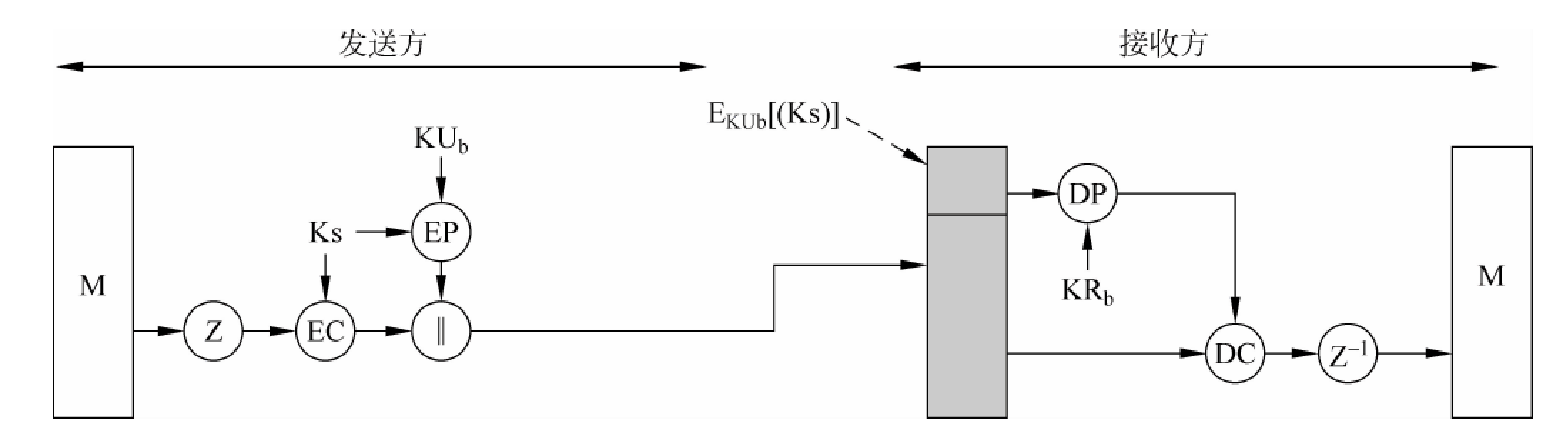


图 9.4 PGP 机密性服务过程

又发现 $E_{KUb}[Ks]$中多了一个),应为 $E_{KUb}[Ks]$。

通过公钥体制保护会话密钥的做法又称数字信封。公钥体制提供“信封”,会话密钥则是“信封”中的内容。

3. 机密性与认证

在 PGP 中,机密性服务和认证服务还可以同时提供。

具体步骤是:首先为明文生成签名并附加到报文首部;然后使用 CAST-128(或 IDEA、3DES)对明文报文和签名进行加密,再使用 RSA 对会话密钥进行加密。

机密性与认证服务过程,如图 9.5 所示。

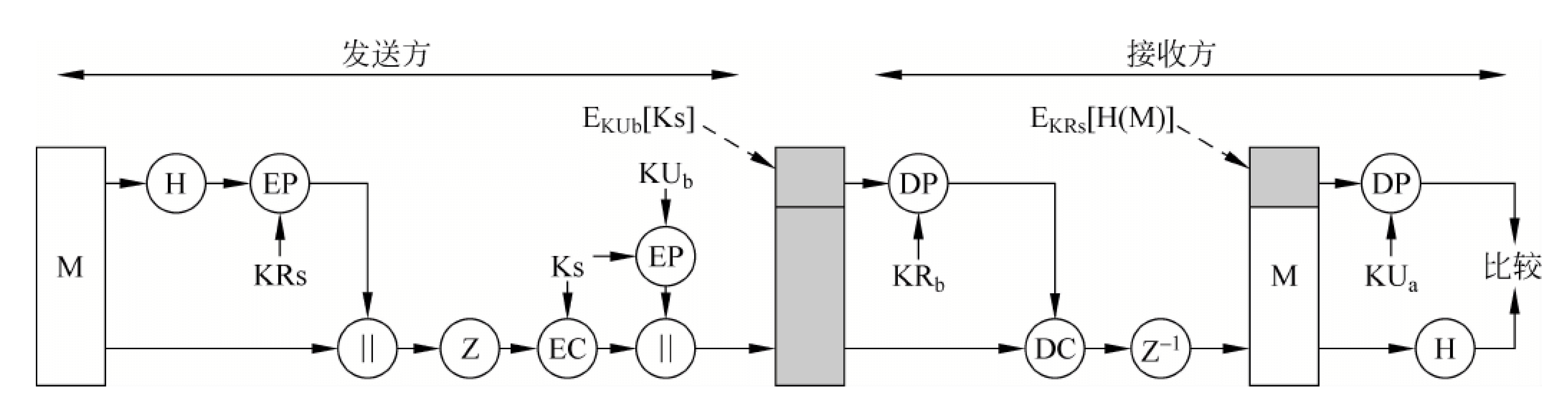


图 9.5 PGP 的机密性与认证服务过程

4. 压缩

PGP 在加密前先要压缩,压缩节省了网络传输的时间和存储空间。PGP 内核使用 PKZIP 算法压缩加密前的明文,对明文的压缩也相当于一次变换,对明文攻击的抵御能力有所提升。

如果考虑签名的因素,压缩的时机一般应在签名后、加密前。

压缩之前生成签名有以下好处:

(1) 验证时直接验证即可,不需要再次重新压缩;

(2) 不必考虑压缩算法的多样性,因为验证与压缩无关。

5. 电子邮件的兼容性

当使用 PGP 时,至少传输报文的一部分需要加密,因此部分或全部的结果报文由任意 8 位字节流组成。由于许多电子邮件系统只允许使用 ASCII 码的正文,所以 PGP 提供了 RADIX-64(即 MIME 的 BASE64 格式)转换方案,将原始二进制流转化为可打印的 ASCII 字符流。

6. 分段与重装

电子邮件还受限于最大报文长度 5000 字节的限制。如果超过这个值,报文将分成报文段,每个段单独发送。

分段在所有其他的过程 (包括 RADIX-64 转换)完成后进行。为提高效率,会话密钥部分和签名部分只在第一个报文段的开始位置出现一次。在接收端,PGP 必须解析当前的电子邮件首部,然后重新装配成原来的报文。

图 9.6 给出了一个 PGP 报文的完整格式。其中,E_{KUb}表示使用用户 b 的公有密钥进行加密;E_{KRa}表示使用用户 a 的私有密钥进行加密;E_{Ks}表示使用会话密钥加密;ZIP 表示 ZIP

压缩功能；R_{64}表示 RADIX-64 转换。

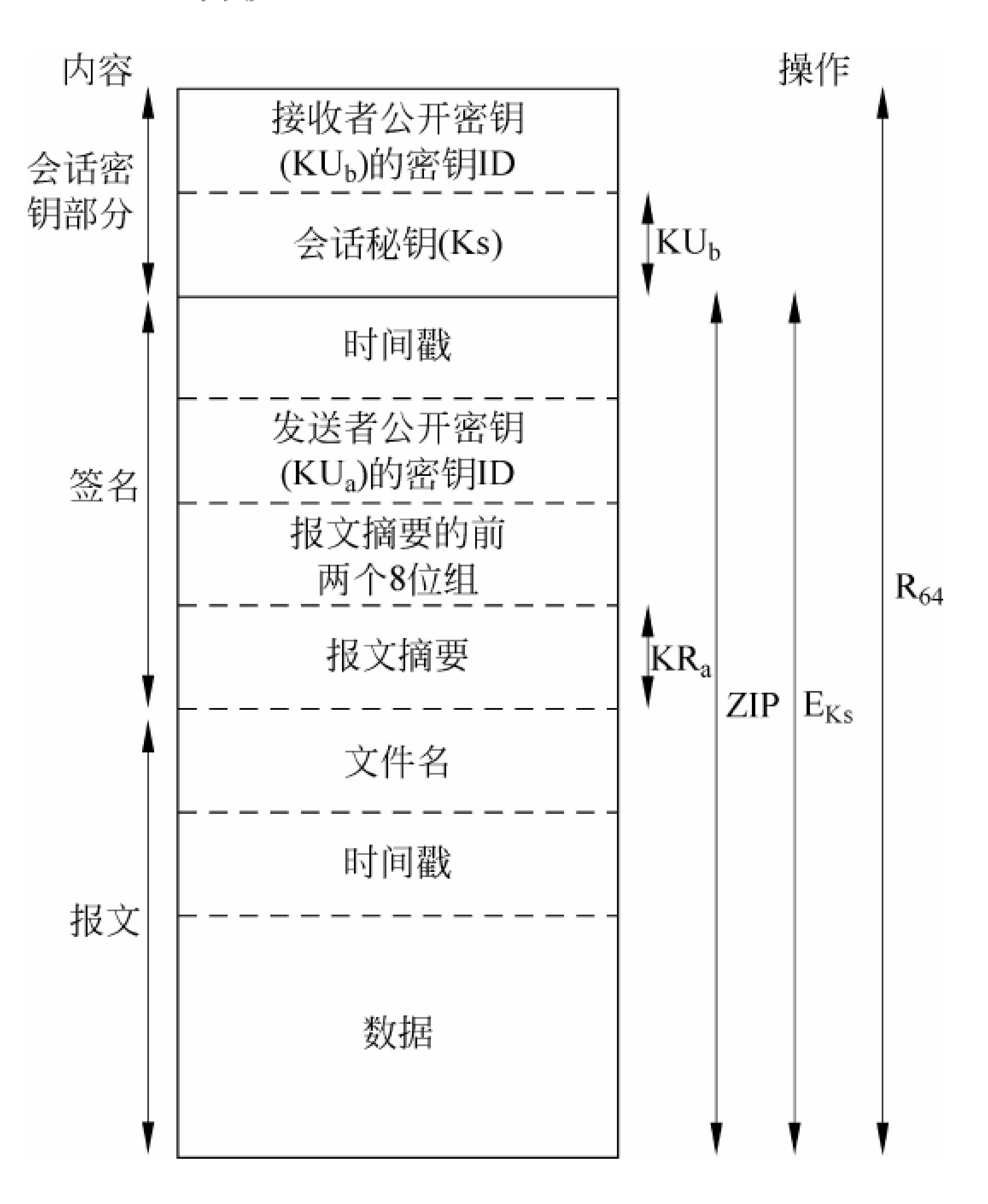


图 9.6 PGP 报文的完整格式

在 PGP 协议中使用了公钥密码体制和对称加密体制，下面介绍相关的密钥管理问题。

PGP 的会话密钥是个随机数，基于 ANSI X9.17 算法由随机数生成器产生。随机数生成器从用户敲键盘的时间间隔上提取随机数种子。磁盘上的种子文件 randseed.bin 采用和邮件加密同样强度的加密算法，能有效地防止攻击者从 randseed.bin 文件中分析出会话密钥产生的规律。

PGP 允许用户拥有多个公钥私钥对，并给每个用户公钥指定一个密钥 ID，由公钥的最低 64 位组成，即：

$$\text{ID} = \text{KU}_a \bmod 2^{64}$$

这个长度使密钥 ID 重复概率非常小。

密钥需要以一种系统化的方法来存储和组织，以便有效和高效地使用。PGP 在每个结点提供一对数据结构，一个用于存储该结点拥有的私钥；另一个用于存储该结点知道的其他所有用户的公钥。相应地，这些数据结构被称为私钥环和公钥环，分别如表 9.1 和表 9.2 所示。

表 9.1 私钥环

时间戳	密钥 ID*	公钥	加密的私钥	用户 ID*
…	…	…	…	…
T_i	$KU_i \bmod 2^{64}$	KU_i	$EH(P_i)[KR_i]$	用户 i
…	…	…	…	…

表 9.2　公钥环

时间戳	密钥 ID*	公钥	可信用户	用户 ID*	合法密钥	签名	可信签名
…	…	…	…	…	…	…	…
T_i	$KU_i \bmod 2^{64}$	KU_i	trust_flag i	用户 i	trust_flag i	—	—
…	…	…	…	…	…	…	…

由于 PGP 主张广泛地在正式或非正式环境下应用，因此没有建立严格的公钥管理模式。

如果 A 的公钥环上有一个从 BBS 上获得的 B 发布的公钥，但已被 C 替换，这时就存在两条通道，即 C 可以向 A 发信并冒充 B 的签名，A 以为是来自 B；A 与 B 的任何加密消息都可能被 C 读取。

为了防止 A 的公钥环上包含错误的公钥，以下方法可用于降低这种风险：

(1) 物理上得到 B 的公钥；

(2) 通过电话验证公钥；

(3) 从双方都信任的实体 D 处获得 B 的公钥；

(4) 从一个信任的 CA 中心得到 B 的公钥(B 的数字证书)。

只要涉及公钥体制，信任问题就不可避免。

PGP 主要有两种信任模型：推荐者信任(或称为间接信任)和公钥信任(或称为直接信任)。

推荐者信任是指相信某推荐者能够做出正确的推荐；公钥信任是指相信某用户与其公钥的对应关系是可靠的。这两种信任都被分为若干个等级，以表示不同的信任程度。

PGP 使用与实体相联系的密钥合法性字段，用来指示 PGP 信任程度，这个字段是由 PGP 从实体的一组签名信任字段中推导出来的。

与 PKI 类似，PGP 信任模型的核心是数字证书。

PGP 证书由以下三部分组成：

(1) 用户 ID。由用户名和其邮箱地址组成，形如：

Ray <Ray@buct.edu.cn>

(2) 公钥。由公钥 ID、公钥数据和创建时间组成。

(3) 若干签名。由一系列对前两项对应关系认同的推荐者的签名组成。

PGP 证书与 X.509 证书有以下区别：

(1) X.509 证书是由专业 CA 颁发，而 PGP 证书则由普通用户颁发。

(2) PGP 使用安全缺陷容忍机制(Security Fault Tolerance Mechanism)，以保证即使在个别推荐者做出错误推荐的情况下仍能够有效运行。

例 9.1　PGP 的应用实例。

安装完成 PGP 以后，打开 PGPKeys，单击 按钮，生成一个新的密钥对，在需要输入密码时输入自己的私钥，其界面如图 9.7 和图 9.8 所示。

当某用户要与其他用户进行通信时，首先要做的是将自己的公钥发给对方。公钥可以通过 Export 导出，如图 9.9 所示。

导出公钥文件后，可以通过各种方式发送给接收方。

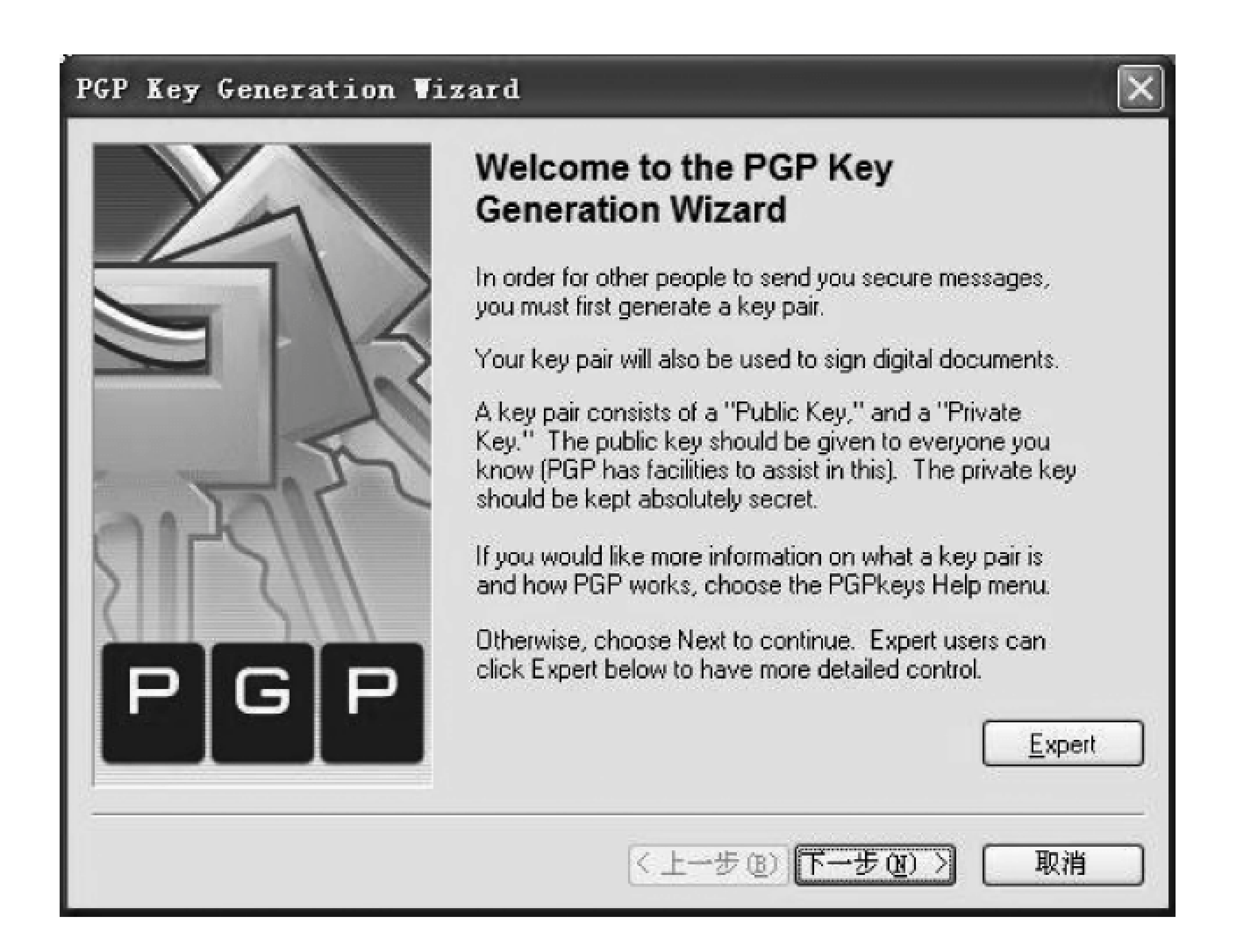


图 9.7　PGP 产生密钥对的界面

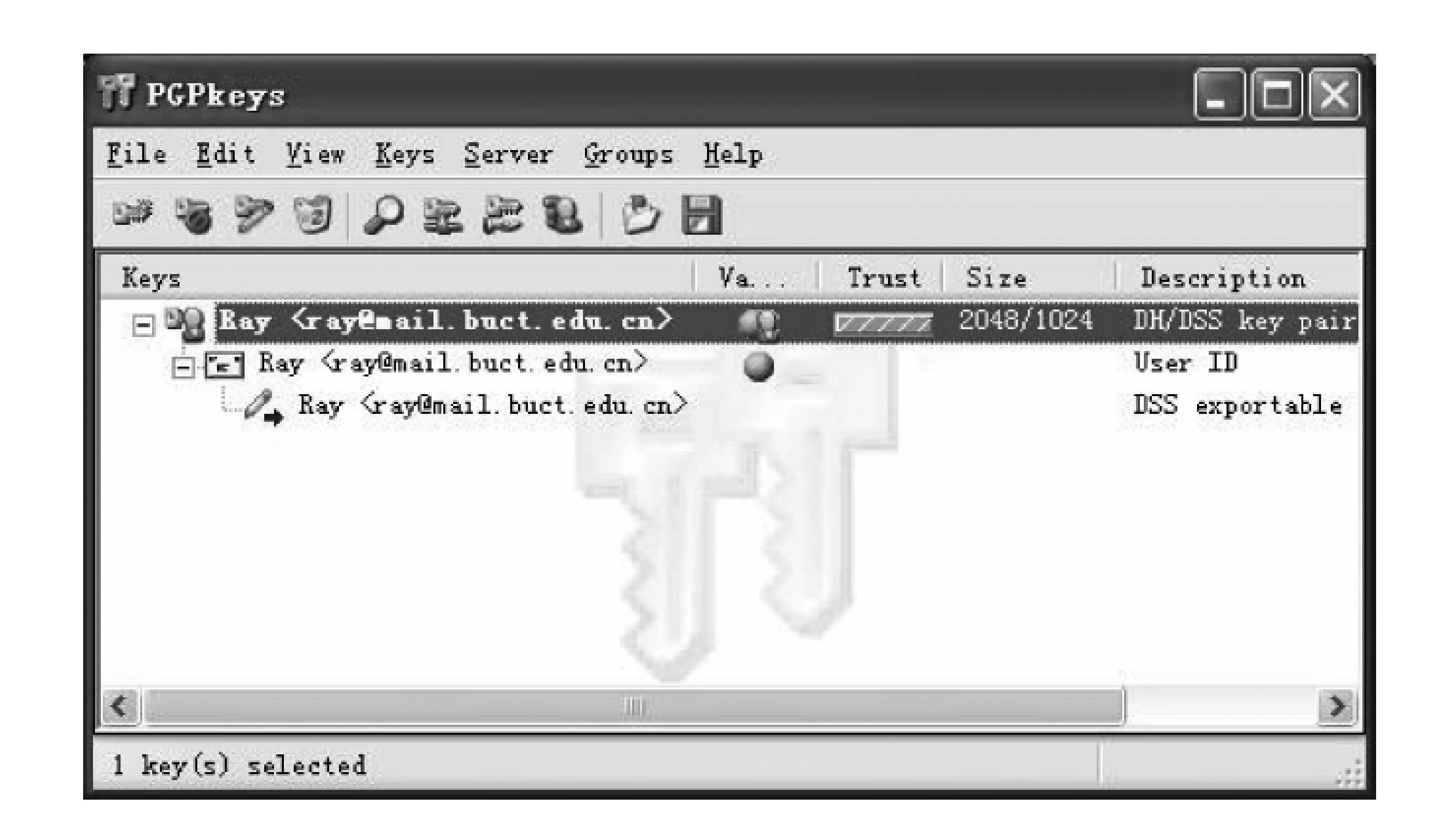


图 9.8　PGP 主界面



图 9.9　导出公钥文件的界面

接收方需要将发送方的公钥导入到自己的公钥环中，导入的方式和导出的方式类似。

如果对文件进行加密和签名，可以右击文件，选择 PGP→Encrypt&Sign 选项，之后用户选择接收方的公钥，然后进行加密并附上自己的签名，如图 9.10 和图 9.11 所示。

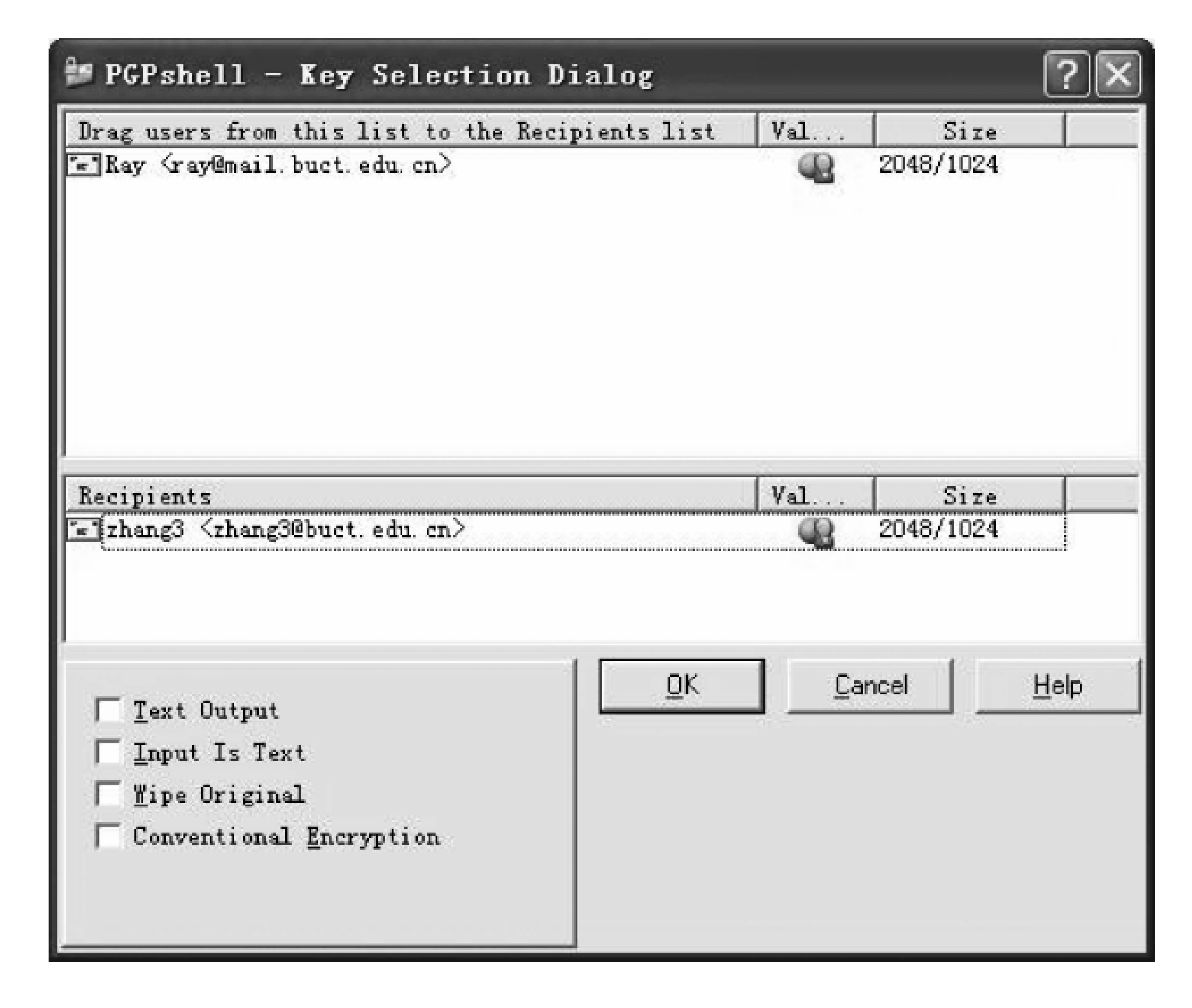


图 9.10　加密界面

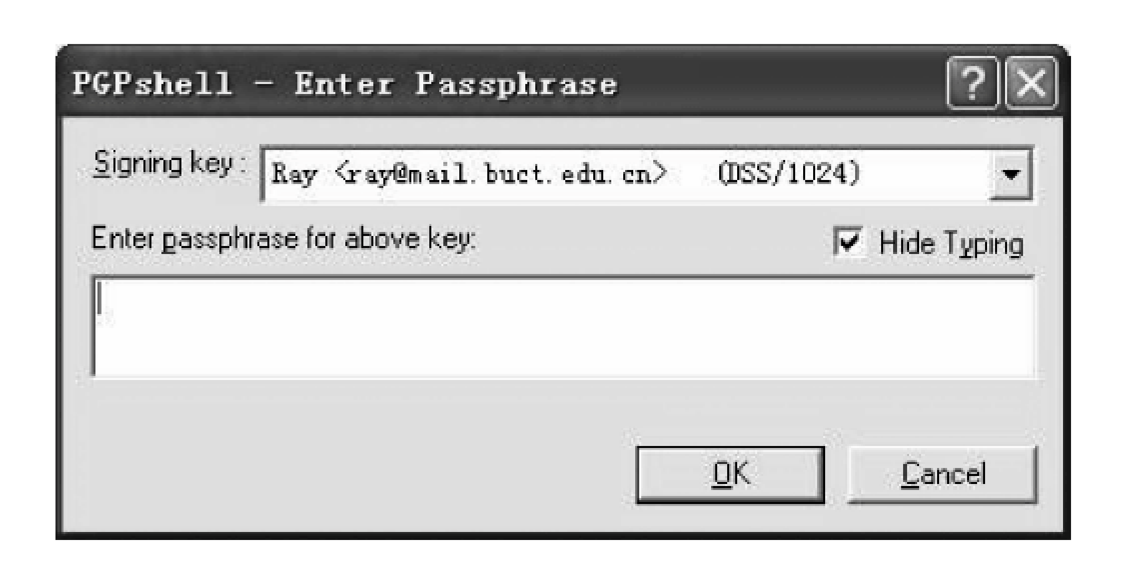


图 9.11　签名界面

签名是用户需要输入自己的密钥值。

用户在接收到文件后，可用自己的私钥解密，并对发送方的签名进行验证，如图 9.12 和图 9.13 所示。

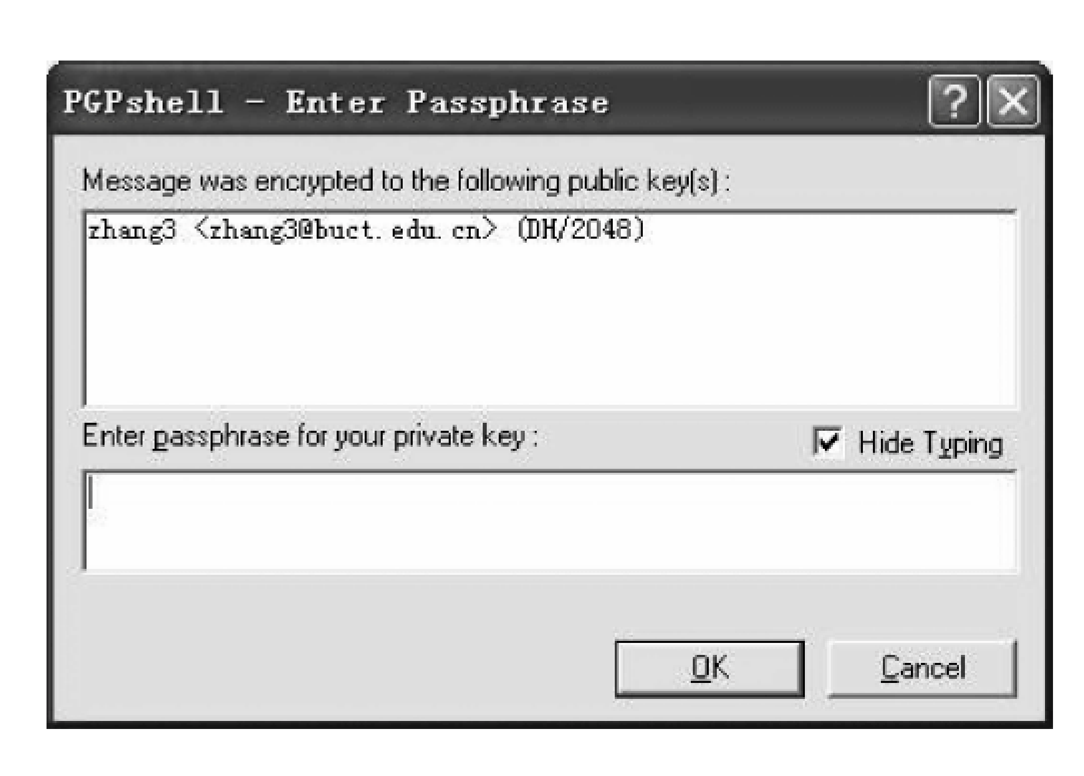


图 9.12　解密界面



图 9.13　验证界面

下面利用 PGP 对电子邮件收发进行加密和签名，在 Outlook Express 中写好邮件，然后单击“加密”和“签名”按钮，如图 9.14 所示。

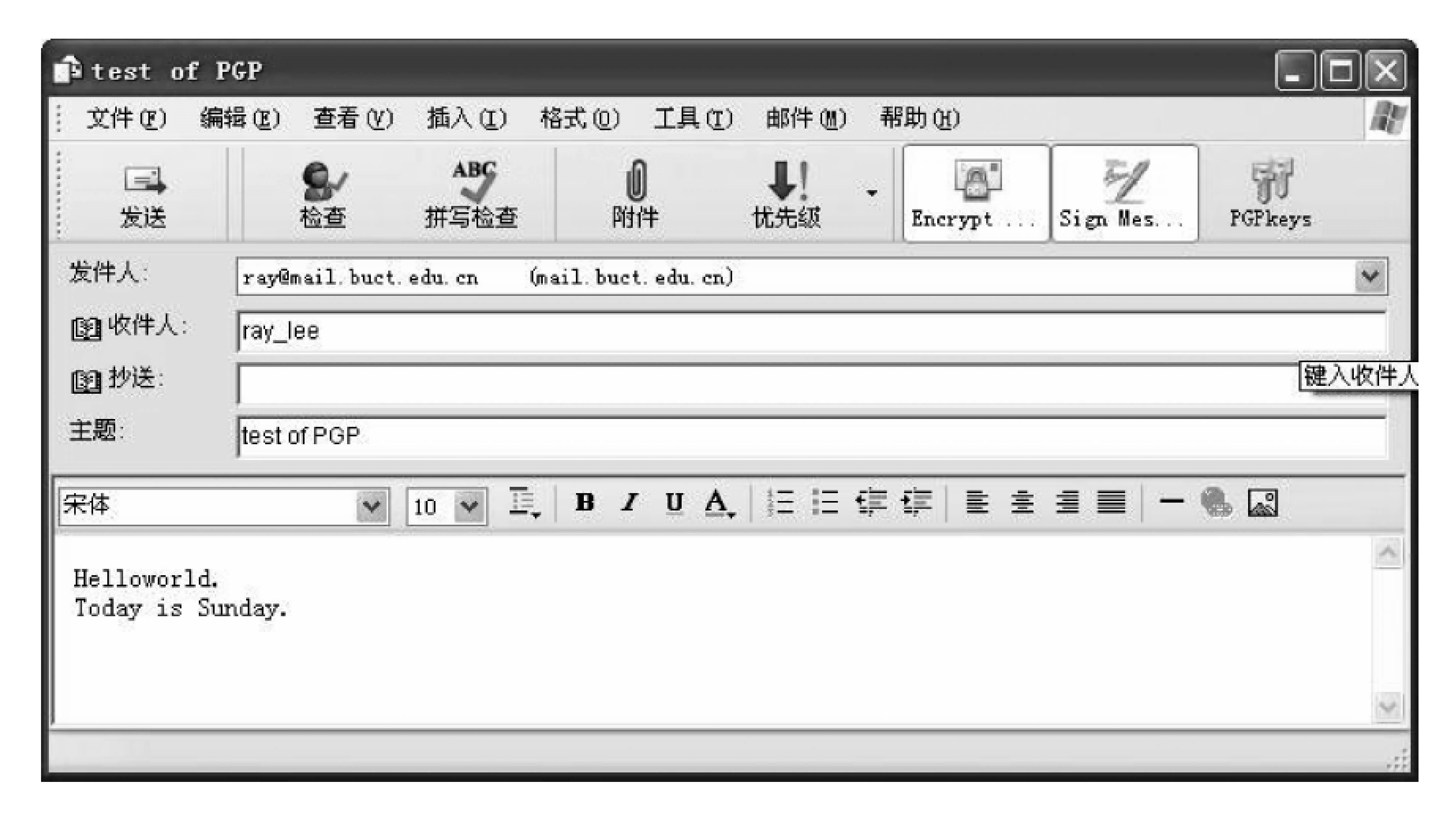


图 9.14　写邮件

因为签名需要用到自己的私钥，所以需要输入密钥值，如图 9.15 所示。

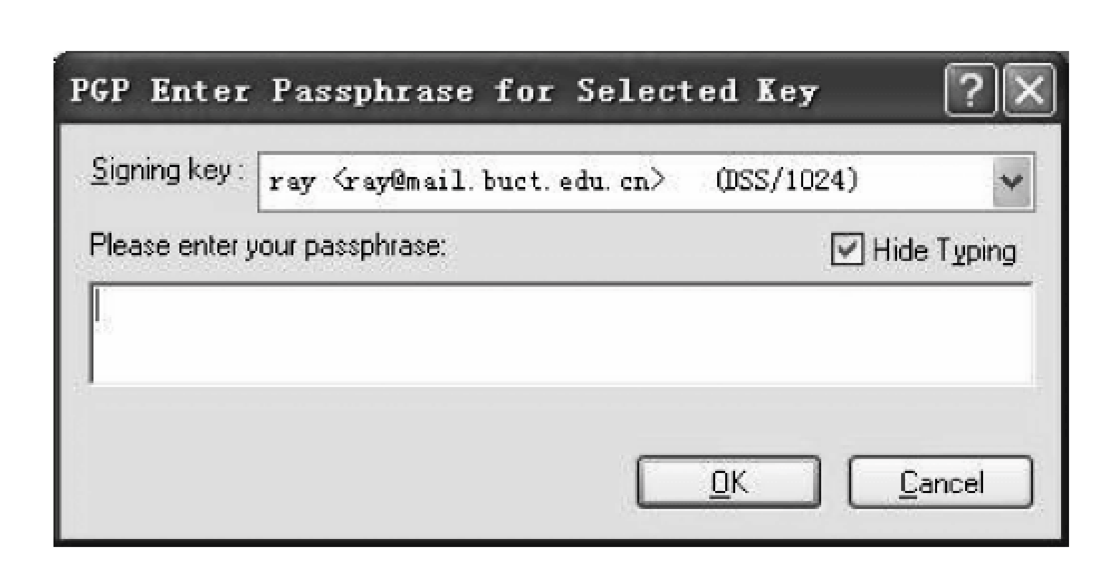


图 9.15　输入签名的密钥值

接收到的邮件如图 9.16 所示，可以看到内容已经变成乱码，不过这些乱码都是可见的字符。

接收方通过 PGP 解密后可以得到邮件的原文，同时能够看到信件的签名信息，如图 9.17 所示。

目前，PGP 的标准化工作进展顺利，相关的 RFC 文档包括以下内容：

(1) RFC1991——PGP Message Exchange Formats。

(2) RFC4880——OpenPGP Message Format。

(3) RFC2015——MIME Security with Pretty Good Privacy (PGP)。

(4) RFC3156——MIME Security with OpenPGP。



图 9.16　加密后的邮件



图 9.17　解密后的邮件

9.3　虚拟专用网

虚拟专用网(Virtual Private Network,VPN)是计算机网络安全中的一个重要技术,通过多种安全机制提供了链路层和网络层上的安全服务。

9.3.1　VPN 基本概念

顾名思义,虚拟专用网可以理解为虚拟出来的企业内部专线。它可以通过特殊的 VPN 协议连接到 Internet 上的位于不同地方的两个或多个企业内部网,建立一条专有的通信线路,就好比是架设了一条专线,但是并不需要真正地去铺设光缆之类的物理线路。一句话,

所谓 VPN,就是利用公共网络建立虚拟私有专用网。

VPN 带来的好处可简单归纳如下:

(1) 降低成本。租用电信的帧中继或 ATM 实现异地的专网也能满足需求,但费用非常昂贵,尤其当分支点比较多时,VPN 则无须申请专线,因此成本较低。

(2) 实现网络安全服务。一般的网络通信是不主动提供安全服务的。建立 VPN 后,可通过加密、访问控制、认证等安全机制来实现网络安全服务。

(3) 自主控制。如果租用电信的专线,那么对线路的控制权在电信一方;而如果使用 VPN,那么对用户认证、访问控制等安全性的控制权就属于 VPN 所有者,这样更容易被用户接受。

VPN 技术本是路由器具有的重要技术之一,目前在交换机、防火墙设备或 Windows 系统中也都支持 VPN 功能。

根据建立 VPN 的目的不同,可以将 VPN 分为以下三类:

(1) 内联网(Intranet VPN)。企业内部虚拟专网与企业内部的 Intranet 相对应。

在 VPN 技术出现以前,公司两个异地机构的局域网想要互联,一般采用租用专线的方式;虽然该方式可以采用隧道等技术,在一端将数据封装后通过专线传输到目的地再解封装。该方式也能提供传输的透明性,但是它与 VPN 技术在安全性上有本质的差异。

利用 VPN 技术可以在 Internet 上组建世界范围内的 Intranet VPN。

(2) 外联网(Extranet VPN)。外联网 VPN 与企业网和相关合作伙伴的企业网所构成的 Extranet 相对应。

这种类型的 VPN 与内联网 VPN 没有本质的区别,只是用于不同公司之间通信,所以需要更多地考虑安全策略的协商等问题。外联网 VPN 的设计目标是既可以向客户、合作伙伴提供有效的信息服务,又可以保证自身的内部网络的安全。

(3) 远程接入(Access VPN),如图 9.18 所示。Access VPN(远程访问虚拟专网)与传统的远程访问网络相对应。

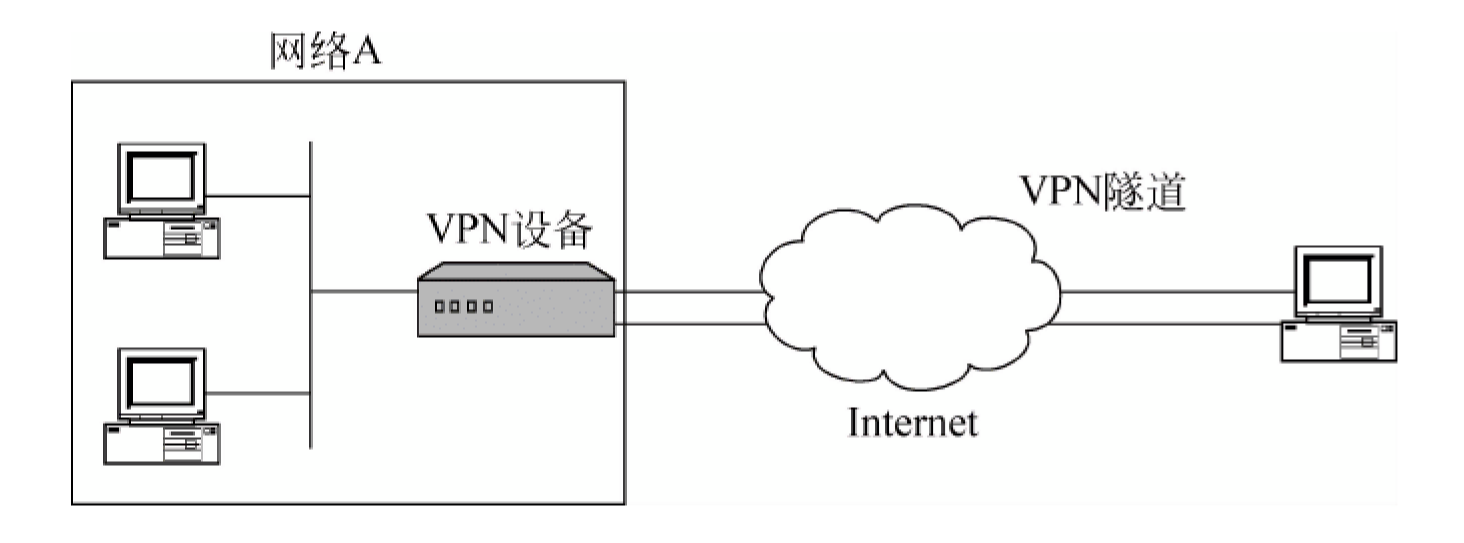


图 9.18　远程接入(Access VPN)

在该方式下远端用户不必像传统的远程网络访问那样,通过长途电话拨号到公司远程接入端口,而是接入到用户本地的 ISP,利用 VPN 系统在公网上建立一个从客户端到网关的安全传输通道。

另一种分类方法是根据网络类型的差异,将 VPN 分为两种类型:Client-LAN 和 LAN-LAN 类型。

显然 Intranet VPN 就是典型的 LAN-LAN 类型,Access VPN 就是典型的 Client-LAN 类型,而 Extranet VPN 既可能是 LAN-LAN 类型也可能是 Client-LAN 类型。

VPN 的基本原理,如图 9.19 所示。

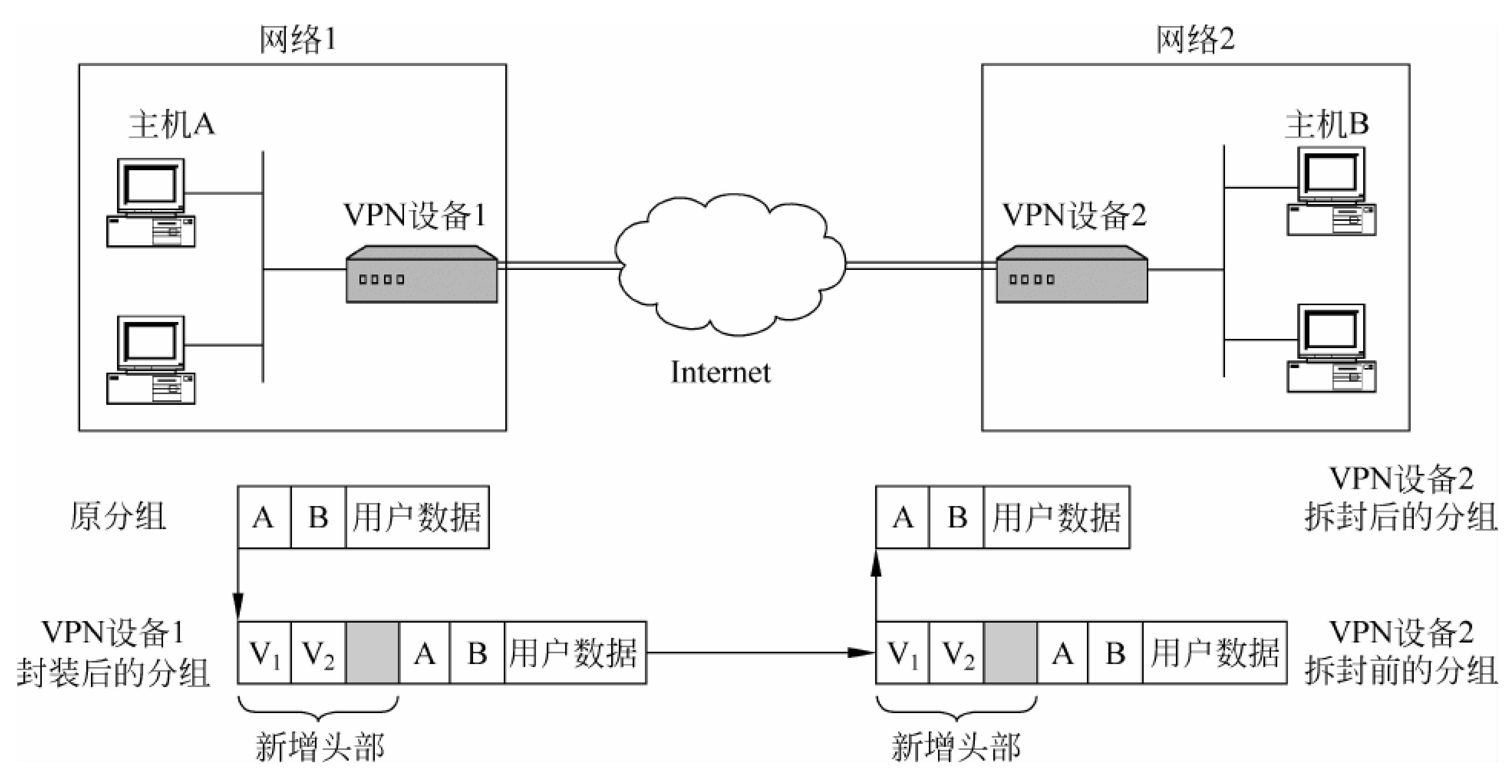


图 9.19　VPN 的基本原理

主机 A 向主机 B 发出类似于企业网内的通信需求时,先由主机 A 建立分组,将其 IP 地址作为源地址,将主机 B 的 IP 地址作为目标地址,将分组发送到 VPN 设备 1 上,通常是 VPN 网关。

分组到达 VPN 设备 1,VPN 设备 1 在分组中增加一新的信息头部。在此分组中,将分组的源 IP 地址设为自己的 IP 地址 V_1,目标地址设为对等 VPN 设备 2 的 IP 地址 V_2,然后发送。

分组通过 Internet 到达 VPN 设备 2,VPN 设备 2 能够识别新增的头部,对其进行拆除,从而得到由主机 A 生成的原分组,然后根据分组的 IP 地址信息,进行类似于企业网内的转发。

9.3.2　隧道与封装

由以上过程可以看出,在构建 VPN 过程中,隧道(Tunneling)技术是 VPN 中的关键技术。

隧道技术是一种通过使用互联网的基础设施在网络之间传递数据的方式。

使用隧道传递的数据(或负载)可以是不同协议的数据包(分组)。隧道协议将这些其他协议的数据包重新封装(Capsule)在新的包头中发送。新的包头提供了路由信息,从而使封装的负载数据能够通过互联网络传递。被封装的数据包在公共互联网上传递时所经过的逻辑路径称为隧道。

隧道的封装协议主要包括以下两类:

一类是二层隧道协议,由于隧道协议封装的是数据链路层的数据包,即 OSI 开放系统互联模型中的第二层的数据包,所以称为第二层隧道协议,如 PPTP、L2F 和 L2TP。二层隧道协议主要用于构建 Client-LAN 型的 VPN。

另一类是三层隧道协议,如 IPSec、GRE 等,它把网络层的各种协议直接封装到隧道协议中进行传输,由于被封装的是 OSI 开放系统互联模型中的第三层网络协议,所以称为第

三层隧道协议。三层隧道协议主要用于构建 LAN-LAN 型的 VPN。

为了理解封装的概念，下面以通用路由封装协议（Generic Routing Encapsulation，GRE）为例来详细说明。

了解 GRE 协议前，先来分析 IP 协议。

IP 包的格式如图 9.20 所示。如果不含选项域，则一般 IP 头长度为 20 个字节。

版本	首部长度	服务类型	总长度（字节）
标识		标志	片偏移
TTL	协议		首部校验和
源 IP 地址			
目的 IP 地址			
选项（可选）			
数据			

图 9.20 IP 包的格式

对于 IPv4 来说，协议版本是 4。服务类型（TOS）域包括一个 3 位的优先权子域，4 位的 TOS 域和 1 位保留位但必须置 0。总长度域是指整个 IP 数据包的长度，以字节为单位。利用首部长度域和总长度域，就可以知道 IP 数据包中数据内容的起始位置和长度。标识域则唯一地标识主机发送的每一份 IP 包，通常每发送一份报文它的值就会加 1。

生存时间域 TTL 设置了数据包可以经过的最多路由器数，它指定了数据包的生存时间。TTL 的初始值由源主机设置，每经过一个处理它的路由器，它的值就减去 1。当该域的值为 0 时，数据包则被丢弃，并发送 ICMP 报文通知源主机。

GRE 通过对某些网络层协议（如 IP 和 IPX）的数据包进行封装，使这些被封装的数据包能够在 IP 中传输。

不妨假设隧道传输采用 IP 协议，被封装数据包采用 IPX 协议，如图 9.21 所示。

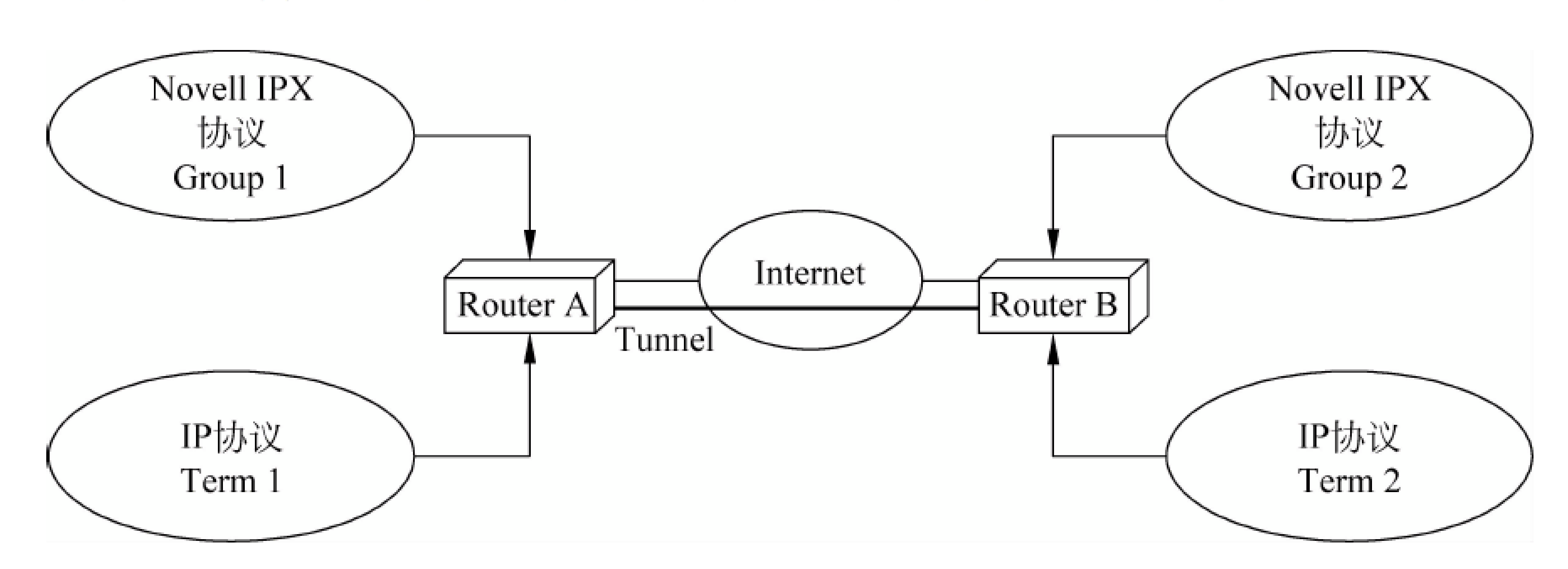


图 9.21 通过 GRE 封装 IPX

数据包要在隧道中传输，必须经过封装与解封装两个过程。

(1) 封装过程。连接 Novell Group 1 的路由器收到 IPX 数据包后，首先 IPX 协议检查 IPX 报头中的目的地址来确定如何路由此包。当发现当前的 IPX 数据包是要发送给 Novell Group 2 时，进行封装。

封装好的报文的形式如图 9.22 所示。IP 协议头是图 9.20 中数据前面的部分，GRE 报文头和 IPX 载荷相当于 IP 包结构的数据部分。

IP 协议头(隧道传输协议)	GRE 协议头(封装协议)	IPX 载荷(被封装协议)

图 9.22　已封装的隧道包格式

(2) 解封装过程。解封装过程和封装的过程相反,Router B 从隧道接口收到的 IP 报文后,通过检查目的地址,发现目的地址恰好是该路由器时,剥掉 IP 协议头,交给 GRE 协议处理(检查校验和报文的序列号等)。GRE 协议处理后,剥掉 GRE 报文头后,再交给 IPX 协议,像对待一般数据包一样对此数据包进行处理。

GRE 报文头结构如图 9.23 所示,各部分的含义如下:

C	R	K	S	s	递归控制位	标志位	版本信息位	协议类型
校验和(可选)							偏移量(可选)	
密钥(可选)								
顺序号(可选)								
路由(可选)								

图 9.23　GRE 报文头结构

(1) GRE 报文头的前 32 位(4 个字节)是 GRE 的基本报头。

其中前 16 位上是 GRE 的标记码:

第 0 位——校验有效位(Checksum Present)

如果置“1”,则校验信息区有效,说明 GRE 报文中校验信息区和分片位移量区都有效。默认置“0”。

第 1 位——路由有效位(Routing Present)。

如果置“1”,则表明分片位移量区和路由区有效;默认置“0”。

第 2 位——密钥有效位(Key Present)。

如果置“1”,表示在 GRE 报文头中密钥信息区有效;默认置“0”。

第 3 位——顺序号有效位(Sequence Number Present)。

如果置“1”,表示顺序号信息区有效;默认置“0”。

第 4 位——严格源路由有效位(Strict Source Route)。

只有在保证所有路由信息均采用严格源路由方式时,该位才置“1”;默认置“0”。

第 5 位——递归控制位(Recursion Control)。

包括 3 位无符号整数,指被允许的附加封装次数;默认设置都为“0”。

5～12 位——保留,目前必须都被置为 0。

13～15 位——保留的版本信息位(Version Number),目前被置为 000。

(2) GRE 报文头的后 16 位是协议类型(Protocol Type),表明有效数据报的协议类型。最基本的是以 IP 协议和 IPX 协议,分别对应的协议号为 0x8000、0x8137。

(3) 下面是可选的 GRE 报文头区(默认则没有)。

校验和(Checksum)16 位:校验和包含 GRE 头和有效分组补充的 IP 校验。如果路由有效位或校验有效位有效,则此区域有效。

偏移量(Offset)16 位:偏移量表示从路由区开始到活动的被检测的源路由入口(Source Route Entry)的第一个字节的偏移量。当且仅当路由有效位或校验有效位有效则

此区域有效。

密钥(Key)32 位：密钥区包含封装操作插入的 32 位二进制数，它可以被接收者用来证实分组的来源。当密钥位有效时此区域有效。

顺序号(Sequence Number)32 位：顺序号包括由封装操作插入的 32 位无符号整数，被接收方用来对那些做了封装的报文建立正确的次序。

(4) 最后是长度不定的路由(Routing)。

以上便是一个完整的 GRE 报文头。正是 GRE 文件头的支持，保证了 GRE 能够有效地完成隧道功能。

有关 GRE 更详细说明可参考 RFC1701 和 RFC1702 文档。

9.3.3 认证协议

VPN 的另一个关键技术是用户认证，用户认证主要由 PPP 功能实现。

点到点协议(Point-to-Point Protocol,PPP)最初的目的是希望通过拨号或专线方式建立点对点连接来交换数据，使其成为各种主机和路由器之间简单连接的通用解决方案。

PPP 协议提供了解决链路建立、维护、拆除、上层协议协商、认证等问题的完整方案，并支持全双工方式。

PPP 包含以下几个部分：

(1) 链路控制协议(Link Control Protocol,LCP)。LCP 负责创建、维护或终止一次物理连接。

(2) 网络控制协议(Network Control Protocol, NCP)。NCP 是一族协议，主要负责确定物理连接上运行的是什么网络协议，并解决上层网络协议出现的问题。

(3) 认证协议。最常用的认证协议是口令验证协议和挑战/握手验证协议。

一个典型的链路建立过程分为以下几个步骤：

(1) 创建 PPP 链路。LCP 负责创建链路，对基本的通信方式进行选择。

具体做法是：链路两端设备通过 LCP 向对方发送配置信息报文(Configure Packets)，一旦一个配置成功信息包(Configure-Ack Packet)被发送且被接收，就完成了交换，进入 LCP 开启状态。

在链路创建阶段，只对验证协议进行选择，用户验证将在下一阶段实现。

(2) 用户验证。在这个阶段，客户端会将自己的身份发送给远端的接入服务器。在认证完成之前，禁止从认证阶段直接进入到网络层协议阶段。如果认证失败，进入链路终止阶段。

在这一阶段中，只有 LCP、认证协议和链路质量监视协议的协议数据是被允许通过，其他的数据将被丢弃。

(3) PPP 回呼控制(Call Back Control)。微软公司设计的 PPP 还包括一个可选的回呼控制协议(CBCP)。

如果配置使用回呼，那么在验证之后远程客户和网络接入服务器(Network Access Server, NAS)之间的连接将被断开，然后由 NAS 使用特定的电话号码回呼远程客户。

这种方法作为一种选择，可以进一步保证拨号网络的安全性。

(4) 调用网络层协议。认证阶段完成之后，PPP 将调用在链路创建阶段选定的各种网络控制协议。选定的 NCP 解决 PPP 链路上的高层协议问题，如在该阶段 IP 控制协议可以

向拨入用户分配动态地址。

经过以上几个过程以后，一条完整的 PPP 链路就建立起来了。

PPP 协议过程状态图如图 9.24 所示。

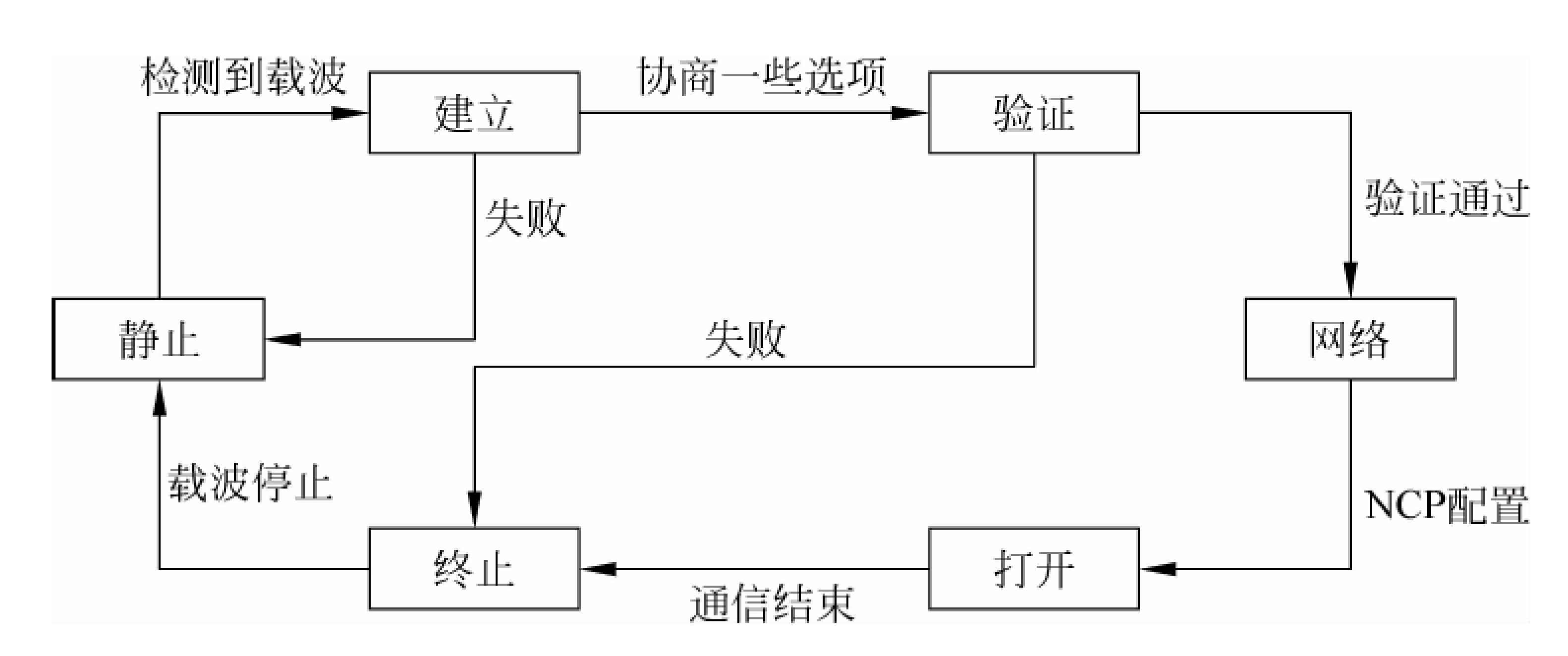


图 9.24 PPP 协议过程状态图

在用户验证过程中，PPP 支持下列认证方式：

(1) 口令验证协议。口令验证协议(Password Authentication Protocol，PAP)是一种简单的明文验证方式，如图 9.25 所示。

NAS 要求用户提供用户名和密码，PAP 以明文方式返回用户信息。显然这种验证方式的安全性较差，攻击者可以很容易地获取被传送的用户名和密码，并利用这些信息与 NAS 建立连接，获取 NAS 提供的有用资源。

(2) 挑战/握手验证协议。挑战/握手验证协议(Challenge-Handshake Authentication Protocol，CHAP)是一种加密的验证方式，能够避免建立连接时传送用户密码的明文，如图 9.26 所示。

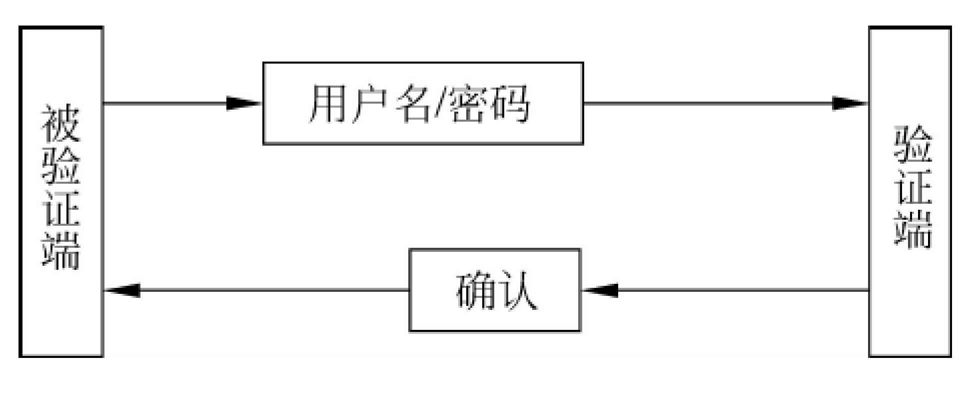


图 9.25 PAP 认证

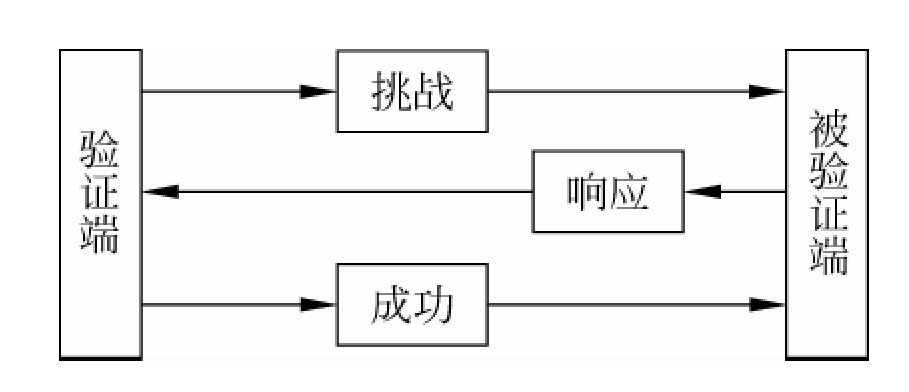


图 9.26 CHAP 验证

NAS 作为验证端，向远程用户发送一个挑战口令，其中包括会话 ID 和一个任意生成的挑战字串。远程客户必须返回用户名、挑战口令、会话 ID 以及用户口令，其中用户名以明文的方式发送，其他信息则是通过 MD5 摘要算法计算出 Hash 值。

CHAP 为每一次验证随机生成一个挑战字串来防止受到攻击(Replay Attack)。在整个连接过程中，CHAP 将不定时地向客户端重复发送挑战口令，从而避免攻击者假冒远程客户进行攻击。

PPP 帧的具体格式如图 9.27 所示。

标志 0x7E 1Byte	地址 0xFF 1Byte	控制 0x03 1Byte	协议 2Byte	数据(<1500 字节)	FCS 2Byte	标志 0x7E 1Byte

图 9.27 PPP 帧格式

PPP 帧的长度都是整数个字节。

PPP 的帧格式前 3 个字段固定为 0x7E、0xFF 和 0x03。

协议字段由两个字节组成。字段中第 8 位必须是 0，最后一位必须为 1。

在“0x0 *** ”到“0x3 *** ”内的协议字段，表示特殊数据包的网络层协议。

在“0x8 *** ” 到“0xb *** ”内的协议字段，表示数据包属于网络控制协议。

在“0x4 *** ”到“0x7 *** ”内的协议字段，表示没有相关 NCP 的低通信量协议。

在“0xc *** ”到“0xf *** ”内的协议字段，表示数据包属于链路层控制协议。

以下的值作为保留：

0xC021	链路控制协议 LCP
0xC023	密码认证协议 PAP
0xC025	链路品质报告 LQR
0xC223	挑战/握手验证协议 CHAP
0x8021	IP 控制协议 IPCP
0x0021	IP 协议
0x0003～0x001F	保留
0x007D	保留
0x00CF	保留
0x00FF	保留
0x8001～0x801F	保留
0x807D	保留
0x80CF	保留
0x80FF	保留

FCS 字段为整个帧的循环冗余校验码，计算范围为 PPP 帧去掉帧头和帧尾标志两个字节的范围。

即使使用所有的帧头字段，PPP 协议帧也只需要 8 个字节就可以完成封装。如果在低速链路上或者带宽需要付费的情况下，PPP 协议允许只使用最基本的字段，将帧头的开销压缩到 2 或 4 个字节的长度，这就是所谓的 PPP 帧头压缩。

一个 LCP 包被封装在 PPP 数据域中，该 PPP 协议域表示为 0xC021。LCP 包的格式如图 9.28 所示。

代码 1Byte	标识符 1Byte	长度 2Byte	数据

图 9.28　LCP 包格式

代码域表示 LCP 包的种类如下。

(1) 链路配置包：用于建立和配置链路。

(2) 链路结束包：用于结束一个链路。

(3) 链路维修包：用于管理和调试一个链路。

标识符在匹配请求和回复中使用。当带有无效标识符域的包被接收时候，该包将被丢弃。

长度域指出 LCP 包的长度，包括代码、标识符、长度和数据域。该长度必须不超过链路的最大接收单元(MRU)，长度域以外的字节被忽略。

数据域是零或多个 8 位字节，由长度域声明。数据域的格式由代码域决定。

有关 LCP 更详细说明可参考 RFC1661 文档。

具体的 PAP 包格式如图 9.29 所示。

PPP 帧头	PPP 协议域	代码	标识符	长度	数据
0x7EFF03	0xC023	1Byte	1Byte	2Byte	

图 9.29 PAP 包格式

代码域代表 PAP 包的类型如下：

(1) 0x01 表示认证请求(Authenticate-Request)；

(2) 0x02 表示认证应答(Authenticate-Ack)；

(3) 0x03 表示认证无应答(Authenticate-Nak)。

标识符用于请求和响应的匹配。

长度域表示 PAP 包的长度，包括代码域、标识符和数据域。超出长度域中长度值的字节将被抛弃。

数据域是零个或多个字节，其格式由代码域决定。

具体的 CHAP 包格式如图 9.30 所示。

PPP 帧头	PPP 协议域	代码	标识符	长度	数据
0x7EFF03	0xC023	1Byte	1Byte	2Byte	

图 9.30 CHAP 包格式

代码域表示 CHAP 包的类型：挑战、应答、成功、失败。

标识符用于挑战、应答和响应的匹配。

长度域表示 CHAP 包的长度，包括代码、长度和数据字段。超出长度的部分将被抛弃。

数据域是零个或多个字节，数据格式由代码域确定。

PPP 协议是目前广域网上应用最广泛的协议之一，具备用户验证能力，可以解决 IP 分配问题。随着宽带以太网的发展，在以太网上运行 PPP 协议来进行用户认证接入的方式成为当前家庭宽带的主流，称为 PPPoE。

PPPoE 是 PPP 协议的一个应用，这里不再赘述。

此外，将 PAP 和 CHAP 进一步扩展还可以得到 PPP 扩展认证协议(EAP)，这种认证协议也逐渐成为主流。

EAP 并不在链路建立阶段指定认证方法，而是把这个过程推迟到验证阶段。于是验证方就可以在得到更多的信息后决定使用什么认证方法。这种机制还允许 PPP 验证方简单地把收到的认证报文传给后方的认证服务器，由后方的认证服务器来真正实现各种认证方法。

后方的认证服务器通常采用 RADIUS(Remote Authentication Dial In User Service)方案。

RADIUS 是为了接入服务器开发的认证系统，它具有集中管理远程访问“拨号用户”的

数据库功能。换句话说，RADIUS是存放用户的“用户名”和“口令”的数据库。接受远程用户访问请求的接入服务器向RADIUS服务器查询该用户是否为合法用户。

有关RADIUS的更详细说明可参考RFC2085文档。

9.3.4 安全关联

VPN的另一个关键技术是安全关联(Security Association，SA)，SA主要用于IPSec型的VPN中。

IPSec提供身份鉴别和信息加密服务。为实现这一目标，通信双方在通信之前必须协商好采用哪种安全协议、加密算法以及相关密钥等问题。所谓的安全关联SA，就是通信双方协商好的安全通信的构建方案，是通信双方共同的约定。

SA是单工的，是从业务流的发送方到接收方的一个单向逻辑关系。在实际应用中的双向的点到点连接中，需要提供两个SA。

安全关联由如下3个参数唯一确定。

(1) 安全参数索引(Security Parameters Index，SPI)：SPI是一个长度为32位的数据，接收方用SPI唯一地确定一个具体的SA。

(2) IP目的地址：即SA中接收方的IP地址。

(3) 安全协议标识符：用以标识通信双方采用的是AH协议还是ESP协议，后面还会介绍。

除以上3个主要参数外，SA还包含以下参数。

(1) 顺序号计数器(Sequence Number Counter)：表示AH或ESP报头中的顺序号。

(2) 顺序号溢出标志(Sequence Counter Overflow)：表示顺序号的溢出。

(3) 防重放窗口(Anti-replay Window)：用来判断AH或ESP数据包是否重放。

(4) AH信息：所采用AH的身份鉴别算法、密钥、密钥生命周期和其他一些相关参数。

(5) ESP信息：所采用ESP的身份鉴别算法、密钥、密钥生命周期和其他一些相关参数。

(6) SA的生命周期：表示在此时间间隔以后，SA结束或者被一个新的SA所替代。

(7) IPSec协议模式：传输模式或隧道模式。

(8) 路径最大传输单元(Path MTU)：指能传输的最大数据包的长度。

AH信息和ESP信息仅当采用AH协议或ESP协议时要求必填外，其他参数在两种协议中都被要求必填。

要建立SA，一般需要使用因特网密钥交换协议(Internet Key Exchange，IKE)(参见RFC2409)。IKE具有一套自保护机制，可以在不安全的网络上安全地分发密钥、认证身份并建立SA。

IKE协商分为以下两个阶段。

第一阶段称为主模式协商。

在网络上建立安全通道，为双方进一步的IKE通信提供机密性、消息完整性以及消息源认证服务。具体包括以下过程：

(1) 策略协商。在这一步中，就4个强制性参数值进行协商：加密算法、Hash算法、认

证方法和 Diffie-Hellman 参数的协商。

(2) 密钥交换。在彼此交换密钥生成种子后,两端主机可以各自生成出完全一样的共享主密钥,保护其后的认证过程。

(3) 认证。密钥交换需要得到进一步认证,如果认证不成功,通信将无法继续下去。主密钥对通信实体和通信信道进行认证。在这一步中,整个待认证的实体数据,包括实体类型、端口号和协议,均由前一步生成的主密钥提供保密性和完整性保证。

第二阶段称为快速模式协商。

主要包括以下过程:

(1) 策略协商。确定使用哪种 IPSec 协议(AH 还是 ESP);确定是否要求加密,如果需要加密,将建立起两个 SA,分别用于入站和出站通信;确定 Hash 算法。

(2) 会话密钥种子的刷新或交换。

(3) SA 和密钥连同 SPI,通过接口传递给 IPSec 驱动程序。

分两个阶段建立 SA 有利于提高密钥交换速度。第一阶段 SA 建立起安全通道后保存在高速缓存中,在此基础上可以建立多个第二阶段 SA 协商,从而提高整个建立 SA 过程的速度。只要第一阶段 SA 不超时,就不必重复第一阶段的协商和认证。允许建立的第二阶段 SA 的个数则由 IPSec 策略决定。

在 VPN 设备上,有一个标准的安全关联数据库(Security Association Database,SAD),其中存放了每一个 SA 的相关参数。

9.3.5 PPTP 协议与 L2TP 协议

基于前面的隧道技术(封装)、认证技术和安全关联技术,接下来将介绍 PPTP 协议与 L2TP 协议。

1. PPTP 协议分析

点对点隧道传送协议(Point-to-Point Tunneling Protocol, PPTP)是由 3Com 和 Microsoft 公司合作开发的第一个用于 VPN 的协议,Windows 中全面支持该协议。

前面介绍过的 PPP、PAP、CHAP 和 GRE 协议构成了 PPTP 协议的基础。

PPTP 的隧道通信包括以下 3 个过程:

(1) PPP 连接和通信。

(2) PPTP 控制连接。它建立到 PPTP 服务器上的连接,并形成一个虚拟隧道。

(3) PPTP 数据隧道。在隧道中 PPTP 协议建立包含加密的 PPP 包的 IP 数据包,这些数据包通过 PPTP 隧道进行收发。

后面的过程取决于前面过程的成功与否。如果有一个过程失败了,那么整个过程都必须重来。

PPTP 协议在一个已存在的 IP 连接上封装 PPP 会话,而不管 IP 连接是如何建立的。也就是说,只要网络层是连通的,就可以运行 PPTP 协议。

PPTP 协议将控制包与数据包分开,控制包采用 TCP 方式传输;数据包部分先封装在 PPP 协议中,然后封装到 GRE 协议中,如图 9.31 所示。

在 PPTP 中,GRE 用于在标准 IP 包中封装协议数据包,因此 PPTP 可以支持多种协议,包括 IP、IPX、NETBEUI 等。



图 9.31 PPTP 协议

PPTP 协议是一个面向中小企业的 VPN 解决方案,它的安全性不算太好,有些场合甚至比 PPP 协议还要弱,因此不适合安全性要求高的场合。

2. L2TP 协议分析

二层隧道协议(Layer 2 Tunneling Protocol,L2TP)将网络层数据包封装在 PPP 帧中,然后通过 IP、X.25、FR 和 ATM 网络中的任何一种点到点串行链路进行传送。

图 9.32 是通过 IP 传送 L2TP 协议数据示意图。

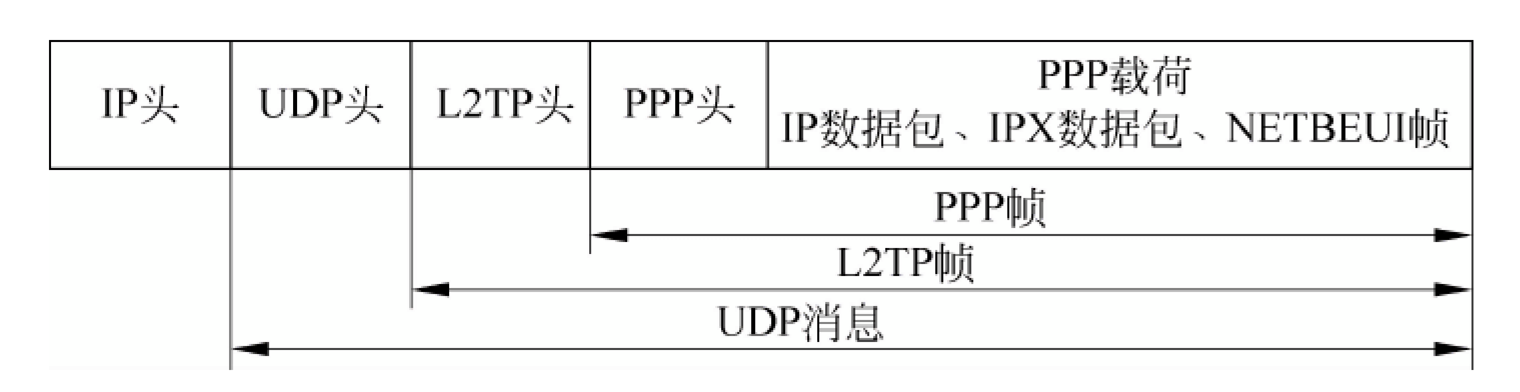


图 9.32 IP 传送 L2TP 协议数据示意图

L2TP 消息有控制消息与数据消息两种类型。控制消息用来建立、保持、清除隧道,数据消息用来封装在隧道中传送的 PPP 帧。

图 9.33 是 L2TP 协议的结构。

PPP 帧	
L2TP 数据消息	L2TP 控制消息
L2TP 数据信道(不可靠)	L2TP 控制信道(可靠)
包传输(UDP、FR、ATM 等)	

图 9.33 L2TP 协议结构

图 9.34 是 L2TP 的头格式。

T|L|x|x|S|x|O|P|x|x|x|x Ver	长度(可选)
隧道 ID	会话 ID
Ns(可选)	Nr(可选)
偏移量(可选)	偏移填充(可选)

图 9.34 L2TP 头格式

L2TP 控制信道与数据信道的包具有相同的头格式。其中:长度、Ns、Nr 对数据消息来说是可选的,但对控制消息则是必需的。

在图 9.34 中,T 表示消息类型,0 为数据消息,1 为控制消息;L 表示长度域的存在,0 表示不存在,1 表示存在;x 保留;S 表示 Ns 与 Nr 域的存在,0 表示不存在,1 表示存在;O 表示

偏移量域的存在，0 表示不存在，1 表示存在；P 表示数据的处理方式；Ver 为 2 时表示 L2TP。

L2TP 定义了以下 4 类控制消息。

（1）控制连接管理消息：

0（保留值）

1（SCCRQ）Start-Control-Connection-request

2（SCCRP）Start-Control-Connection-Reply

3（SCCCN）Start-Control-Connection-Connected

4（StopCCN）Stop-Control-Connection-Notification

5（保留值）

6（HELL）Hello

（2）调用管理消息：

7（OCRQ）Outgoing-Call-Request

8（OCRP）Outgoing-Call-Reply

9（OCCN）Outgoing-Call-Connected

10（ICRQ）Incoming-Call-Request

11（ICRP）Incoming-Call-Reply

12（ICCN）Incoming-Call-Connected

13（保留值）

14（CDN）Call-Disconnect-Notify

（3）错误报告消息：

15（WEN）WAN-Error-Notify

（4）PPP 会话控制消息：

16（SLI）Set-Link-Info

L2TP 的控制连接协议对隧道的建立、保持、认证和清除进行了规定，可参考 RFC2661 文档。

尽管 PPTP 和 L2TP 都使用 PPP 协议对数据进行封装，然后添加附加包头用于数据在互联网上的传输，但实质上它们之间有很大的差别。

（1）PPTP 要求传输网络为 IP 网络。L2TP 对传输网络的要求不高，可以在 IP（使用 UDP）、帧中继永久虚拟电路（PVCs）、X. 25 虚电路以及 ATM 虚电路上使用，也就是说 L2TP 只要求提供面向数据包的点对点连接即可。

（2）PPTP 只能在两点之间建立单一隧道，L2TP 允许在两点间建立多个隧道。使用 L2TP 用户可以针对不同的服务质量创建不同的隧道，这是很有用的功能，而 PPTP 不具备这一功能。

（3）L2TP 可以提供包头压缩。当压缩包头时，系统开销仅占用 4 个字节，而 PPTP 协议必须占用 6 个字节。

（4）L2TP 可以提供隧道验证，而 PPTP 则不支持。

例 9.2　Windows XP 下配置 PPTP/L2TP 客户端实例。

配置过程如下：

在“网络邻居”图标上右击，在弹出的快捷菜单中选择“属性”选项，在打开的界面中单击

“创建一个新的连接”,再在网络连接界面中选中“连接到我的工作场所的网络”单选按钮,然后根据网络连接向导建立 VPN 的连接(客户端),如图 9.35 所示。

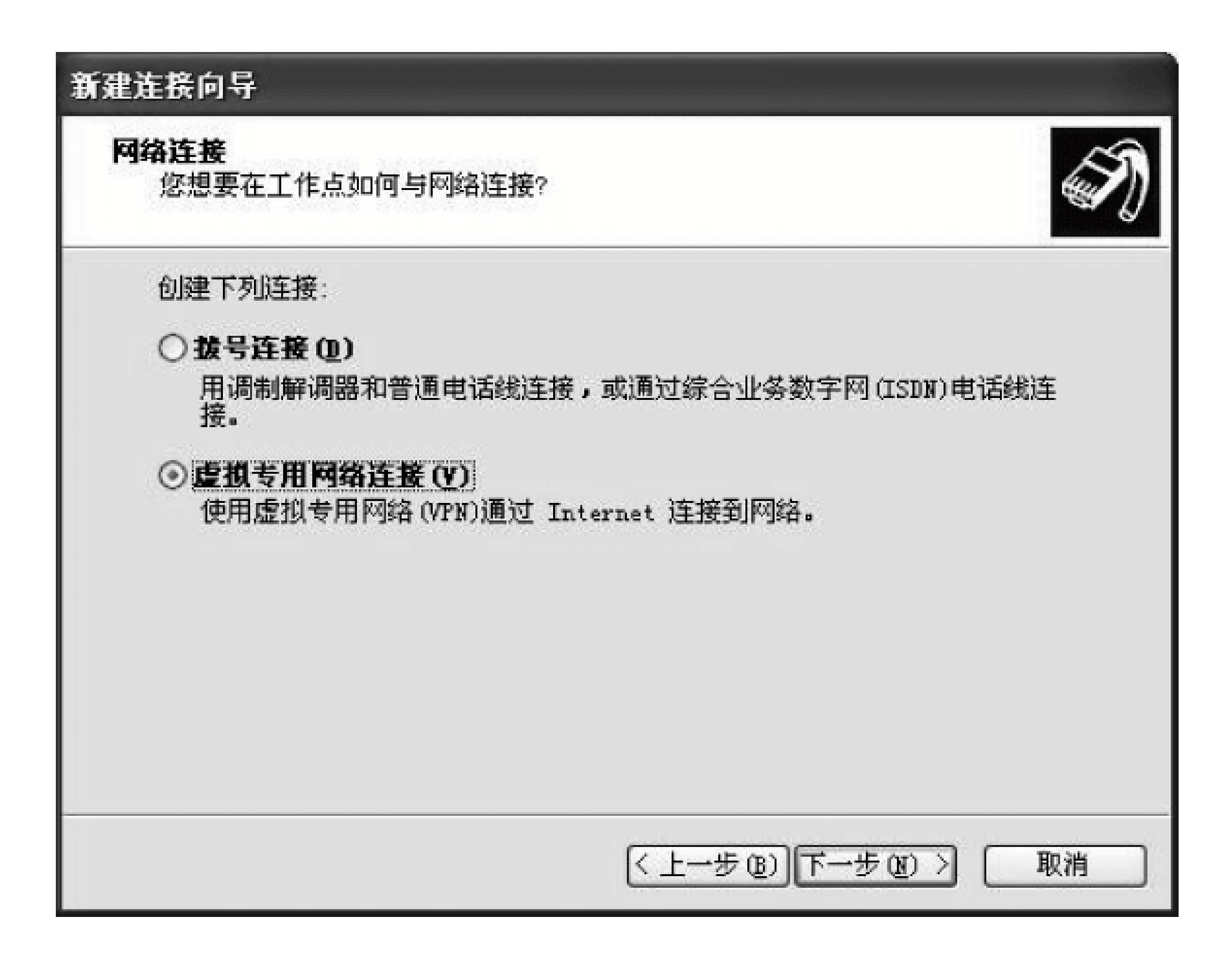


图 9.35 用网络连接向导建立 VPN 连接

右击已建立的虚拟专用网络图标,选择“属性”选项,可详细配置 VPN 的属性。

如图 9.36 所示,可以选择隧道协议类型,Windows XP 支持 PPTP 和 L2TP 两种隧道协议,也可以设置成自动选择隧道协议。

也可以设置数据加密类型和身份认证协议,如图 9.37 所示。





图 9.36 选择隧道协议类型



图 9.37 数据加密和身份验证协议设置

此外,对该 VPN 连接还可设置以下参数:

(1) 目的主机名或目的 IP 地址。

(2) 重拨参数(重拨次数、重拨间隔、挂断前的空闲时间、断线是否重拨)。

(3) 是否提示名称、密码和证书。

(4) 是否此连接共享。

(5) 是否启用 LCP 扩展。

(6) 是否启用软件压缩。

(7) 是否为单链路连接协商多重链接。

例 9.3　Windows 2000 Server 下配置 PPTP/L2TP 服务器。

配置过程如下：

执行“程序”→“管理工具”→“路由和远程访问”命令，在“本地属性”中选择“远程访问服务器”选项可以建立 PPTP/L2TP 服务器。

图 9.38 是配置隧道类型的窗口，Windows 2000 Server 支持 GRE、PPTP 和 L2TP 等隧道类型。

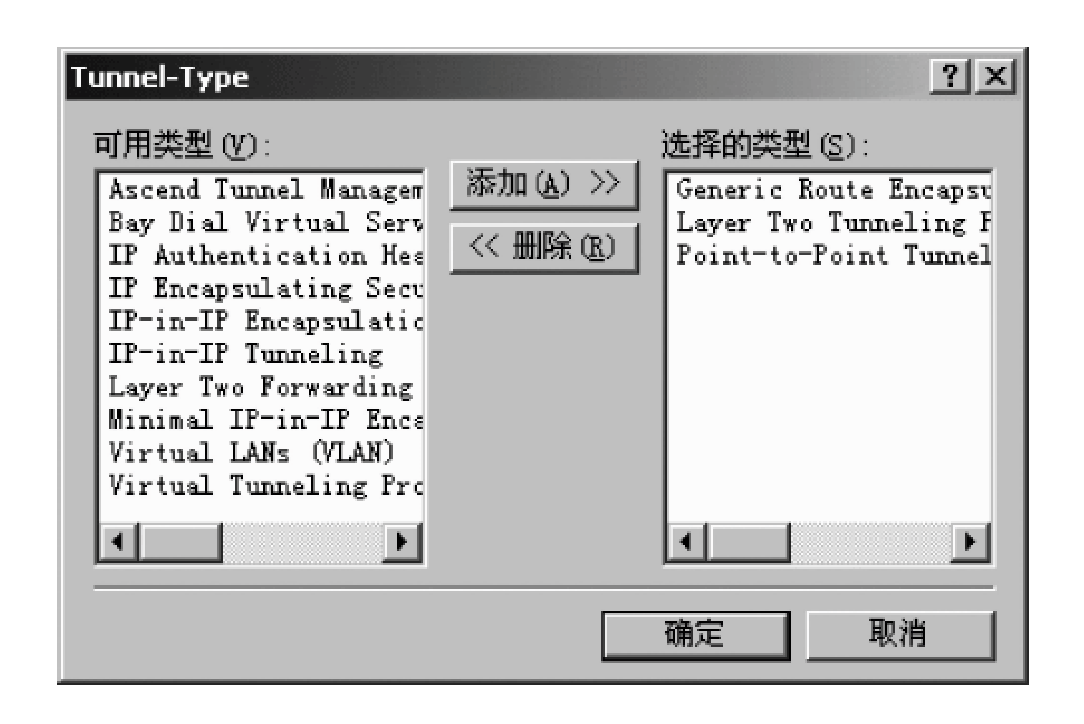


图 9.38　隧道类型窗口

图 9.39 是对身份验证、加密等安全性选项的设置，Windows 2000 Server 支持 MS-CHAP、CHAP、PAP 等多种认证方式。

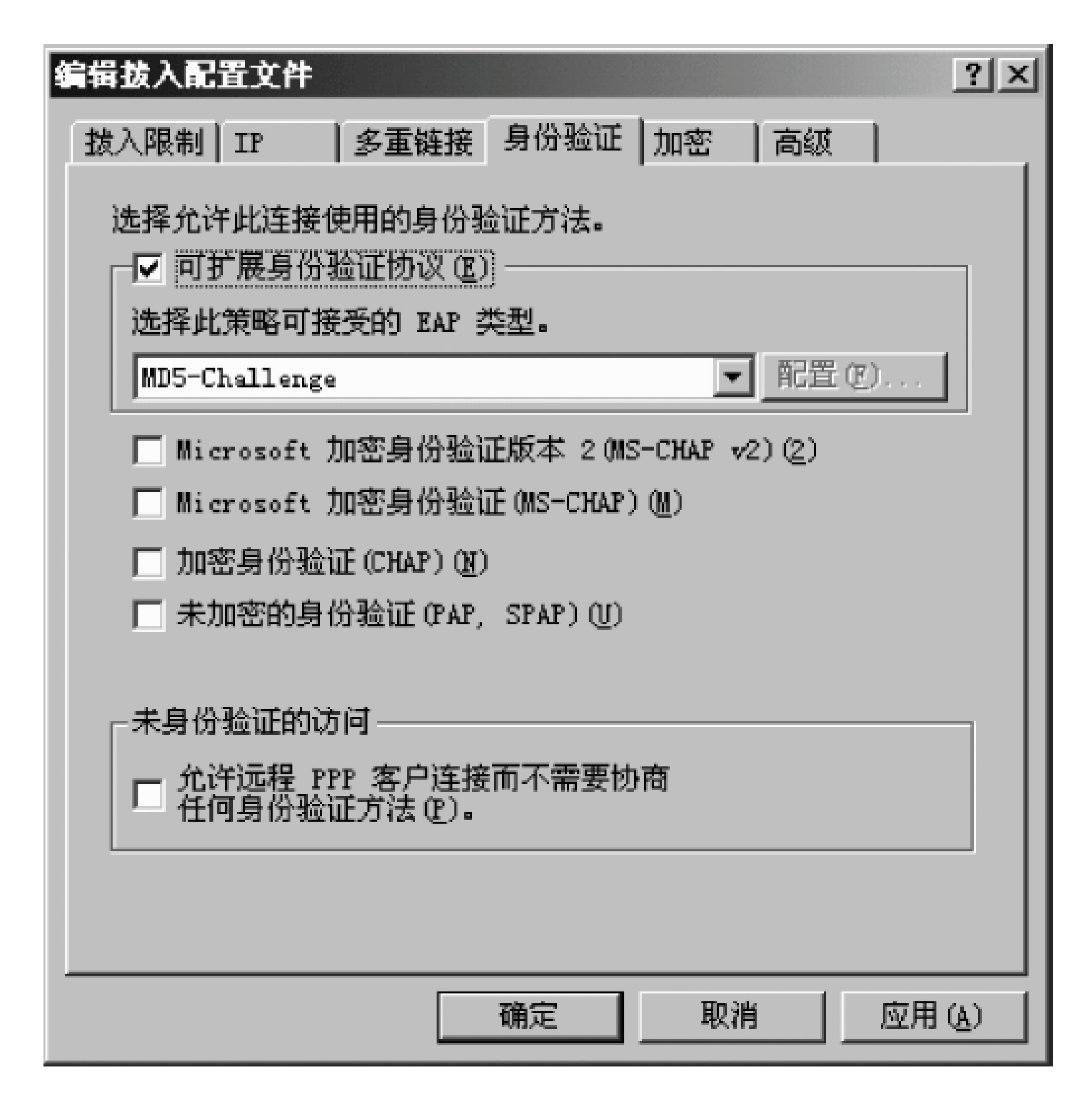


图 9.39　身份验证、加密等选项的设置

默认状态下，Windows 2000 Server 支持的 VPN 端口数是 5 个，必要时端口数量还可以进行调整。

9.3.6 IPSec 协议

IPSec 是 IETF 为保证在 Internet 上传送数据的安全保密性能而制定的三层隧道加密协议，在 IP 层对 IP 报文提供安全服务。该协议中定义了如何在 IP 数据包中增加字段来提供安全机制：数据源认证、完整性保护、机密性保护和防重放保护等。

IPSec 提供了两个主机之间、两个安全网关之间或主机和安全网关之间的保护。

所谓安全网关，是指实现 IPSec 协议的一个中间系统。例如，一个实现了 IPSec 的路由器、防火墙或一台服务器就是一个安全网关。

IPSec 协议主要包括 AH 协议、ESP 协议两个协议。

AH 协议可提供数据源认证和数据完整性校验功能；ESP 协议除可提供数据认证和完整性校验功能外，还提供对 IP 报文的加密功能。

IPSec 有隧道(Tunnel)和传输(Transport)两种工作模式。传输模式只能适合 PC 到 PC 的场景，而隧道模式可用于任何情形。

1. 认证头(AH)

认证头(Authentication Header，AH)由下一头、净荷长度、保留值、安全参数索引(SPI)、序列号、认证数据(可变)6 个域组成，如图 9.40 所示。

下一头	净荷长度	保留值
安全参数索引(SPI)		
序列号		
认证数据(可变)		

图 9.40 认证头格式

这些域都是强制性的。

AH 既支持传输模式又支持隧道模式，如图 9.41 所示。



图 9.41 传输模式和隧道模式

传输模式通常在两个主机之间的 VPN 上使用，使用原始明文 IP 头，仅对数据进行加密，当然也包括它的 TCP 和 UDP 头。

隧道模式通常在安全网关之间使用。隧道模式处理整个 IP 数据包，包括全部 TCP 或 UDP 数据，它用自己的地址作为源地址加入到新的 IP 头中。

在传输模式中，AH 被插入在 IP 头之后，在上层协议(如 TCP、UDP、ICMP 等)之前，或在任意已经插入的 IPSec 头之前。

在隧道模式中，AH 保护整个内 IP 包，包括整个内 IP 头。

认证数据由 MD5 和 SHA-1 算法生成，用来保证数据完整性。

2. 封装安全净荷(ESP)

封装安全净荷(Encapsulating Security Payload,ESP)用于提供混合安全服务,ESP 为 IP 数据包提供的安全服务有机密性保护、数据源认证、完整性保护和防重放保护。此外,ESP 的隧道模式还支持对报文路径信息的隐藏。

ESP 可以单独使用,也可以与 AH 组合使用,或者以嵌套的方式在隧道模式中使用。

与 AH 类似,ESP 的安全服务可以在一对正在通信的主机之间提供,可以在正在通信的安全网关之间提供,或者可以在主机与安全网关之间提供。

ESP 包由安全参数索引(SPI)、序列号、数据(可变)、填充(0～255 字节)、填充长度、下一头和认证数据(可变)共 7 个域组成,如图 9.42 所示。

安全参数索引(SPI)		
序列号		
数据(可变)		
填充(0～255 字节)	填充长度	下一头
认证数据(可变)		

图 9.42 封装安全净荷包格式

在这些域中,填充域和认证数据域是可选的,其他的域都是必选的,可选域是否选取由 SA 指定。

在传输模式下,ESP 被插入在 IP 头之后,在上层协议(如 TCP、UDP、ICMP 等)之前,或在已经插入的 IPSec 头之前。

在隧道模式下 ESP 的位置,相对于外 IP 头与在传输模式中 ESP 的位置一样。

加密算法也由 SA 指定,可使用对称加密算法加密。

ESP 必须实现的算法包括 CBC 模式的 AES、用 MD5 的 HMAC 和用 SHA-1 的 HMAC。

表 9.3 概括并比较了 AH 和 ESP 在两种模式下的功能。表 9.4 概括并比较了 AH 和 ESP 的安全服务。

表 9.3 传输模式与隧道模式中 AH 和 ESP 功能的比较

认证服务方式	传输模式 SA	隧道模式 SA
AH	认证 IP 数据和 IP 头中的一部分	认证整个内部 IP 包和部分外部 IP 包头部分
不带认证的 ESP	加密 IP 数据	加密内部 IP 包
带认证的 ESP	加密 IP 数据、认证 IP 数据	加密内部 IP 包、认证内部 IP 包

表 9.4 AH 和 ESP 的安全服务

安全服务	AH	不带认证的 ESP	带认证的 ESP
访问控制	√	√	√
完整性	√		√
数据源认证	√		√
防重放保护	√	√	√
保密性		√	√
有限通信业务流保密性		√	√

防重放保护是AH和ESP都提供的安全服务，下面对防重放保护的原理进行说明。

这里的重放攻击是指攻击者在得到一个已经认证过的包后，反复将其传送到目的站点的行为。当重复接收经认证的包发生时，没有采取针对性措施的应用程序将可能出现异常，导致服务中断或其他不可预料的后果。

AH和ESP的序列号域的使用可防止这种重放攻击。

当一个新的SA建立时，发送方将序列号初始值置0，每次在SA上发送一个包时，计数器就会加1并将新值填入序列号域。如果不考虑防重放需求，序列号可以循环使用，即当序列号达到2^{32}后，重新置0，周而复始。

为了实现防重放，SA放弃循环计数机制，即当序列号达到$2^{32}-1$时，SA终止，后续的通信将在新的SA上进行。

VPN一个很重要的优势在于对网络层的透明性。

9.4 安全套接层

VPN从二层或三层解决了安全传输的问题，有了VPN，公司的各个分支机构就可以跨地域地连接起来了。不过，无论如何VPN的维护是需要较大开销的，尤其是在隧道模式下。接下来要介绍的安全套接层(Secure Socket Layer，SSL)，相对来说是一种更容易实施的安全传输方案。

9.4.1 SSL的体系结构

SSL最早是由Netscape公司开发出来的，后来逐渐形成事实上的业内标准。

尽管SSL也是一种提供安全服务的手段，但SSL还是与前面介绍的VPN有本质的差别，这是因为SSL提供的是在TCP上的安全服务。

SSL协议可用于保护正常运行于TCP之上的任何应用，如HTTP、FTP、SMTP或Telnet的通信，最常见的是用SSL来保护HTTP的通信。

SSL由两层结构——处于相对底层的SSL记录协议与处于相对高层的SSL握手协议、修改密码协议和报警协议组成，如图9.43所示。

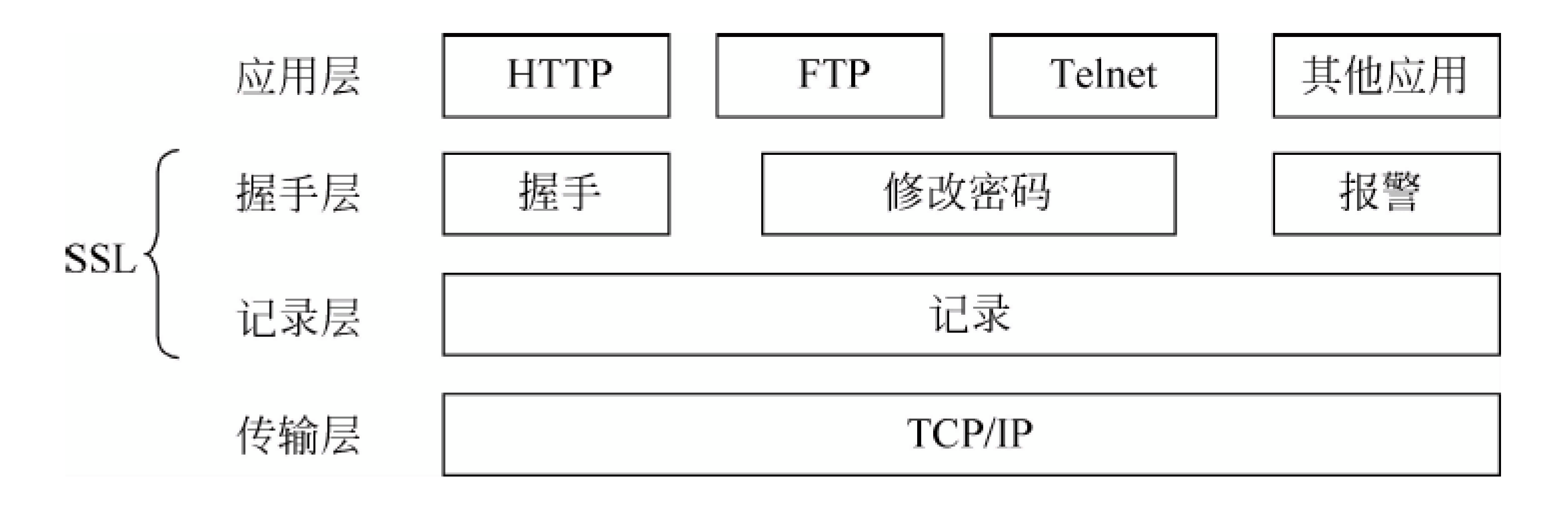


图9.43 SSL的层次

SSL提供的面向连接的安全性具有以下3个基本性质：

(1) 数据是加密的：在初始握手确定密钥后，用对称加密算法加密数据。

(2) 实体的身份能够用证书或公钥进行认证。

(3) 连接是可靠的：消息的可靠传输通过安全 Hash 函数及 MAC 来保证。

SSL 会话是状态化的，存在连接状态和会话状态。所有的记录都是在会话状态下进行处理，连接状态的安全参数由 SSL 握手协议设置。握手协议能够有选择地将连接状态转化为会话状态。

SSL 会话状态记录以下元素。

(1) 会话标识符：标识会话状态的字节序列。

(2) 实体证书：实体的 X.509 证书，也可设为空。

(3) 压缩方法：加密前压缩数据的算法。

(4) 密码说明：指定加密算法和 MAC 算法。

(5) 主密钥：客户端与服务器之间的共享秘密(48B)。

(6) 可恢复标志：一个会话是否能够用来发起新的连接的标志。

SSL 连接状态包括以下元素。

(1) 服务器与客户随机数：客户方与服务器方提供的随机数。

(2) 服务器写 MAC 密码：服务器写数据时用于对数据进行 MAC 操作的密码。

(3) 客户端写 MAC 密码：客户端写数据时用于对数据进行 MAC 操作的密码。

(4) 服务器写密钥：服务器加密数据、客户解密数据的密钥。

(5) 客户端写密钥：客户端加密数据、服务器解密数据的密钥。

(6) 初始向量：进行 CBC 模式加密时使用的数据。

(7) 序列号：每个连接状态都包含一个序列号，并且是读、写分离的。

9.4.2　记录协议

SSL 记录协议用于封装上层协议。从高层接收未解释的任意长度的非空数据块，负责对其进行分段、压缩、解压缩、净荷保护处理。

(1) 分段。为了便于处理，需要将上层数据分成若干个段，每个段小于或等于 2^{14}B。

(2) 压缩。为了减少网络带宽的消耗，每段数据都将被压缩。为了将来能够完全恢复出来，压缩算法必须采用无损压缩算法。压缩是可选的过程，但是如果选择压缩，当压缩效果不好时，应避免压缩后增加的长度少于 1024B(注意：并不是在所有情况下，压缩都会使数据被压缩后长度变短)。

(3) 增加 MAC。接下来需要对压缩后的数据计算 MAC，MAC 被添加在压缩数据之后。

(4) 加密。对压缩数据连同 MAC 使用对称加密算法进行加密，加密造成长度的增加也应该少于 1024B。

可选的加密算法有 IDEA、RC4-128、DES、3DES 等。

(5) 添加 SSL 记录头。SSL 协议的最后一步是在加密数据前添加 SSL 头。

SSL 头由以下部分组成：

① 内容类型，封装的高层协议类型，8 位。

② 主版本，SSL 的主版本，如 3(SSL3.0)，8 位。

③ 次版本,SSL 的次版本,如 0(SSL3.0),8 位。

④ 压缩长度,压缩数据的字节长度,16 位。

9.4.3 修改密码协议和报警协议

修改密码协议仅包含一条消息,该消息由一个值为 1 的字节构成。

修改密码协议说明消息可由客户端或服务器发送,通知接收方后面的记录将由新协商的密码提供保护。发送方发送此消息后,立即要求记录层把即将写状态变成当前写状态;接收方收到此消息后,立即要求记录层把即将读状态变成当前读状态。

报警协议传达消息的严重性并描述报警,致命错误的报警信息可能导致连接立即终止。与其他消息一样,报警消息同样被加密和压缩。

致命错误的报警消息包括以下内容。

(1) MAC 记录错误消息:接收到的 MAC 非法。

(2) 握手失败消息。

(3) 解压失败消息:压缩数据不正确导致解压不能成功。

其他的报警消息还包括以下内容。

(1) 结束通知消息:发送者通知接收者不再用此连接发送任何消息。

(2) 不支持的证书消息。

(3) 证书期满消息。

(4) 证书撤销消息。

(5) 未知证书消息。

报警协议由两个字节组成:第一个字节描述报警级别,1 表示警告,2 表示致命错误;第二个字节包含描述特定报警消息的代码。

9.4.4 握手协议

SSL 握手协议允许客户端与服务器为记录层取得一致的安全参数,进行相互认证、协商安全参数和报警约定。

SSL 握手协议在 SSL 记录层上面执行操作,产生会话状态所需的密码参数。当 SSL 客户和服务器要开始通信时,他们确认协议版本,选择密码算法,相互认证(可选),并利用公钥密码体制产生会话密钥。

图 9.44 描述了握手过程,握手过程包括客户端和服务器之间交换一系列的消息。每个消息包含 3 个消息域,分别是消息类型、长度和内容。

(1) 消息类型。握手协议的消息类型包括 Hello Request、Client Hello、Server Hello、Certifcate、Server Key Exchange、Certificate Request、Server Done、Certificate Verify、Client Key Exchange、Finished,占 1 个字节。

(2) 长度。消息的长度以字节为单位,占 3 个字节。

(3) 内容。

以 Client Hello 消息为例,消息相关参数如下。

（1）版本：客户支持的最高 SSL 版本。

（2）随机数：此随机数用于在密钥交换中防止重放攻击。

（3）会话标识：非 0 表示希望更新连接的参数；0 表示希望创建一个新连接。

（4）密码组：客户端支持的密码算法列表。

（5）压缩算法：客户端支持的压缩算法列表。

SSL 握手协议包括下列步骤：

（1）交换 Hello 消息，以确认算法、交换随机值并为重开会话进行检测。

（2）交换必要的密码参数，使客户端与服务器能确认预主密钥。

（3）交换证书和密码信息，使客户端和服务器相互认证。

（4）从交换的随机值计算出密钥。

（5）向记录层提供安全参数。

（6）允许客户端和服务器验证他们的对等实体，确认握手是在没有攻击者篡改的情况下发生。

在实际应用中，SSL 协议的运算需要消耗大量资源，为此一些厂家通过硬件实现 SSL 协议，这样可大大提高 SSL 的效率，称为“SSL 加速器”。

严格地说，SSL 并非互联网上的开放标准，因为其中涉及不少专利技术。于是，IETF 组织开发了 TLS，用来替代 SSL，不过 TLS 与 SSL 非常相似。当然，TLS 与 SSL 之间也存在一些不同之处，如 TLS 主要采用非专利的加密算法，并且能够提供更全面的报警消息。

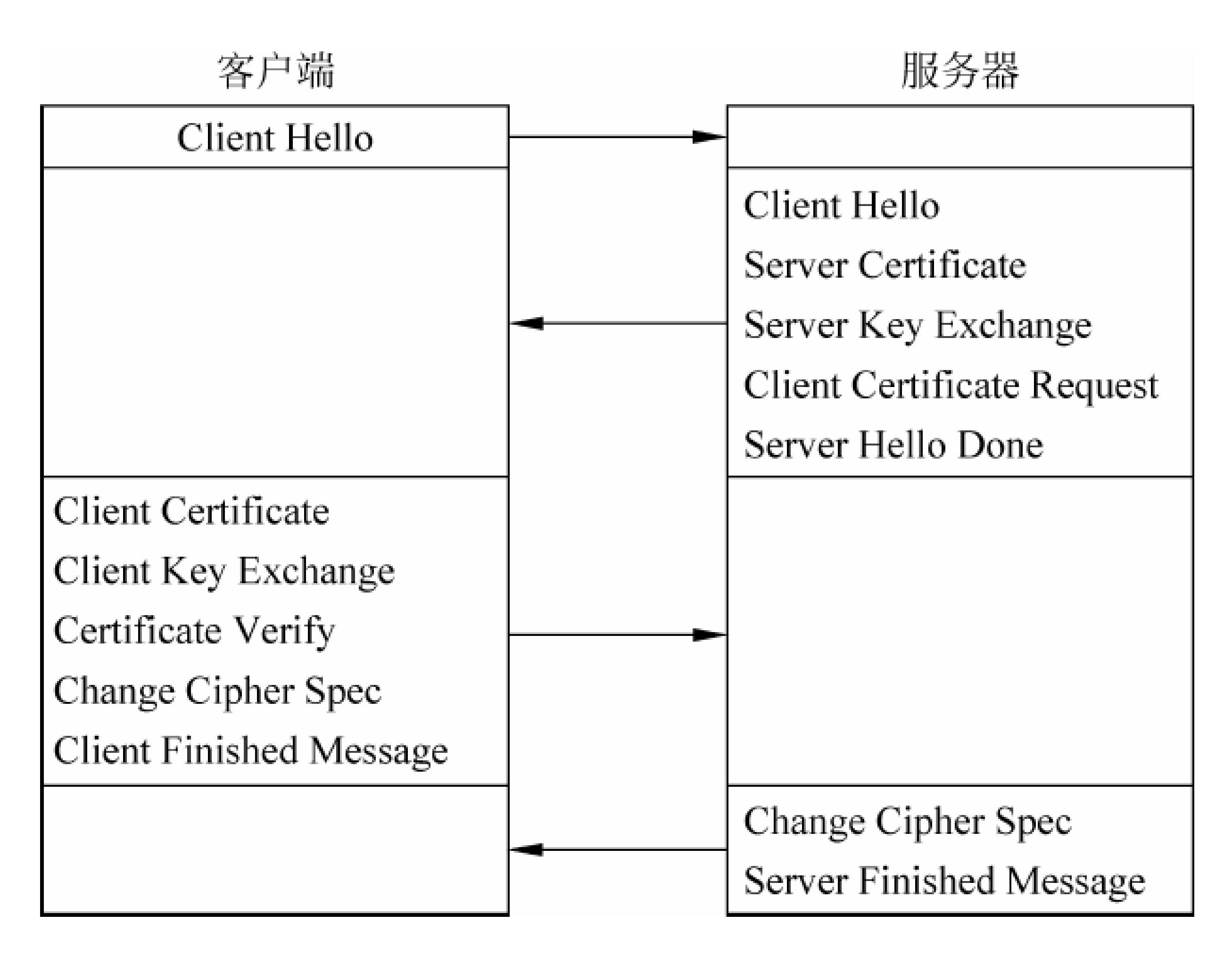


图 9.44　握手过程

9.4.5　SSL 应用

SSL/TLS 通常使用两种办法为应用提供安全服务，一种是仿照 SSL/TLS 协议的功能，对应用协议进行升级，如 RFC2487 的 SMTP over TLS 协议；另一种是为实施 SSL/TLS 保护的应用协议重新分配一个端口，对应用层的透明性使得这种方式应用更为广泛。

表 9.5 给出了基于 SSL/TLS 的一些应用协议端口分配情况，其中的端口号都是由 IANA 规定的。

表 9.5 基于 SSL(或 TLS)的应用协议端口

服务名	端口号	描 述
NSIIOPS	261/TCP	SSL(或 TLS)上的 IIOP 名字服务
HTTPS	443/TCP	SSL(或 TLS)上的 HTTP 协议
SMTPS	465/TCP	SSL(或 TLS)上的 SMTP 协议
NNTPS	563/TCP	SSL(或 TLS)上的 NNTP 协议
SSHELL	614/TCP	SSL 上的 SHELL
LDAPS	636/TCP	SSL(或 TLS)上的 LDAP 协议
FTPS-DATA	989/TCP	SSL(或 TLS)上的 FTP 协议和数据
FTPS	990/TCP	SSL(或 TLS)上的 FTP 控制
Telnets	992/TCP	SSL(或 TLS)上的 Telnet 协议
IMAPS	993/TCP	SSL(或 TLS)上的 IMAP4 协议
IRCS	994/TCP	SSL(或 TLS)上的 IRC 协议
POP3S	995/TCP	SSL(或 TLS)上的 POP3 协议

例 9.4 Windows 系统中的 Outlook Express 和 Internet Explorer 都可以设置成为 SSL 的客户端。图 9.45 是 Outlook Express 的 SSL 设置，图 9.46 是 Internet Explorer 的 SSL 设置。

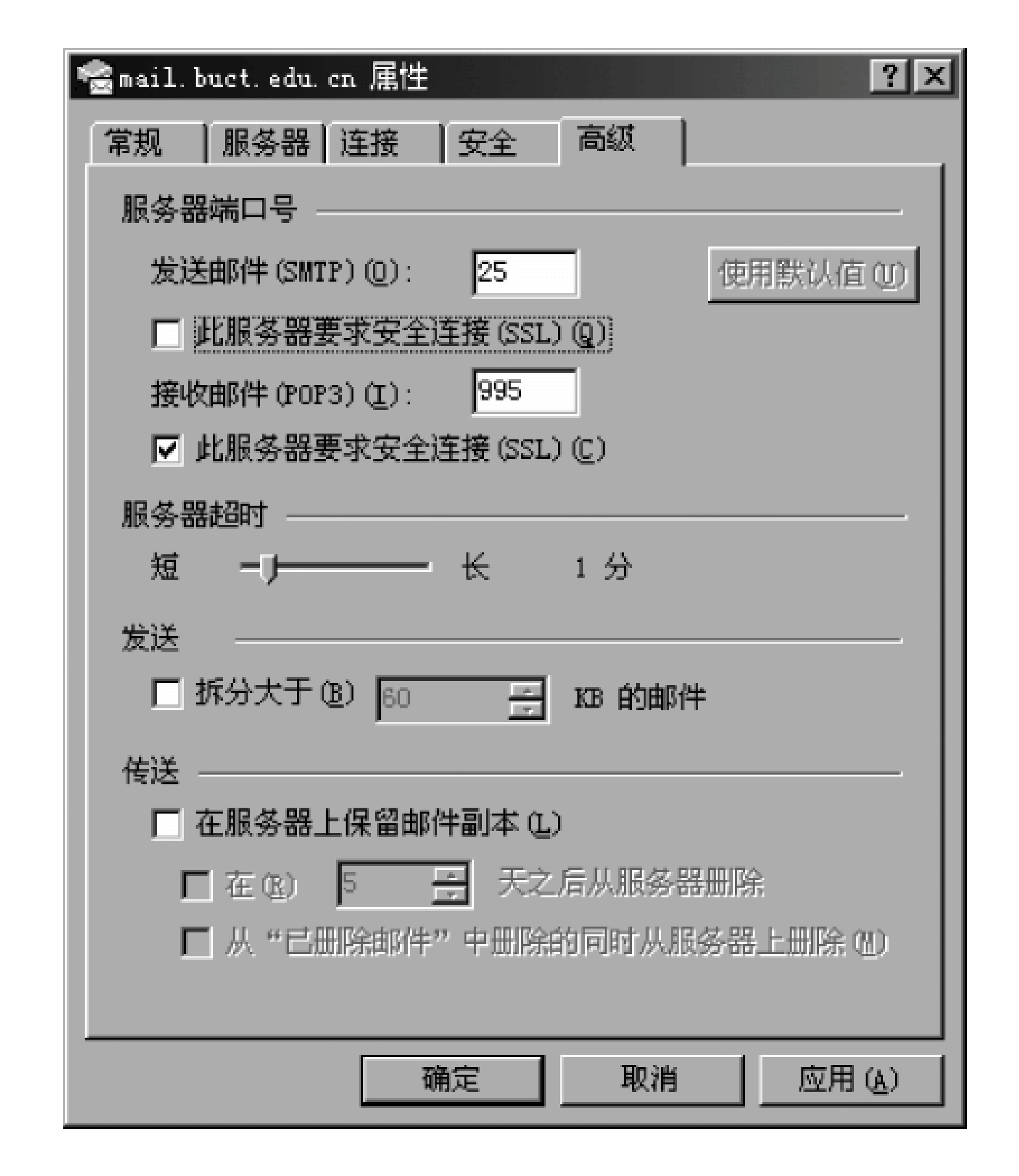


图 9.45 在 Outlook Express 中配置 SSL 属性



图 9.46 在 Internet Explorer 中配置 SSL 属性

9.5 安全电子交易

通过引入 SSL 可以帮助一些应用实现安全通信，如基于 SSL 的 HTTP、SMTP、FTP 等。但是应用的种类复杂多样，仅仅依靠 SSL 并不能完全满足应用层安全的需要。例如，

如何在不保证安全的网络上实现安全的电子商务就是一个复杂的课题。安全电子交易协议是目前公认的较好的电子商务解决方案。

9.5.1　SET 协议

安全电子交易(Security Electronic Transaction,SET)是一种开放的安全协议和标准,用于在互联网上实现安全的交易活动。

SET 最初是由 Visa 和 MasterCard 合作开发完成的,参与该规范设计的公司还包括 IBM、Microsoft、Netscape、VeriSign 等。

SET 是以信用卡支付为基础的网上电子支付系统规范,通常 SET 包括以下一些参与者,其示意图,如图 9.47 所示。

(1) 持卡人(Cardholder)。持卡人持有发卡机构发行的、经过授权的支付卡,如 MasterCard 和 Visa 卡。在互联网中,持卡人与商家通过电子交易的方式实现购买和支付关系。

(2) 商家(Merchant)。商家为持卡人提供丰富的商品,提供商品的方式主要是 Web 网站或 E-mail 方式。商家在商家银行里设立了账号,用来与支付卡的发卡机构进行划账交易。

(3) 发卡机构(Issue)。发卡机构通常是为持卡人提供支付卡的金融机构,如银行,持卡人可直接到发卡银行办理支付卡。当持卡人在商家处有购买行为时,发卡机构向商家银行划账。

(4) 商家银行(Acquirer)。商家银行是指为商家建立账户的金融机构,用来接收支付卡的付款。商家银行还可以向商家提供一定的认证机制,确保持卡人的卡具有合法性并且卡上余额足以支付商品。当持卡人的资金划到商家银行后,商家银行还向商家提供已支付的凭证。

(5) 认证中心(CA 机构)。SET 协议中,需要认证中心提供相互间实体的认证。认证中心为持卡人、商家、发卡机构、商家银行提供证书。

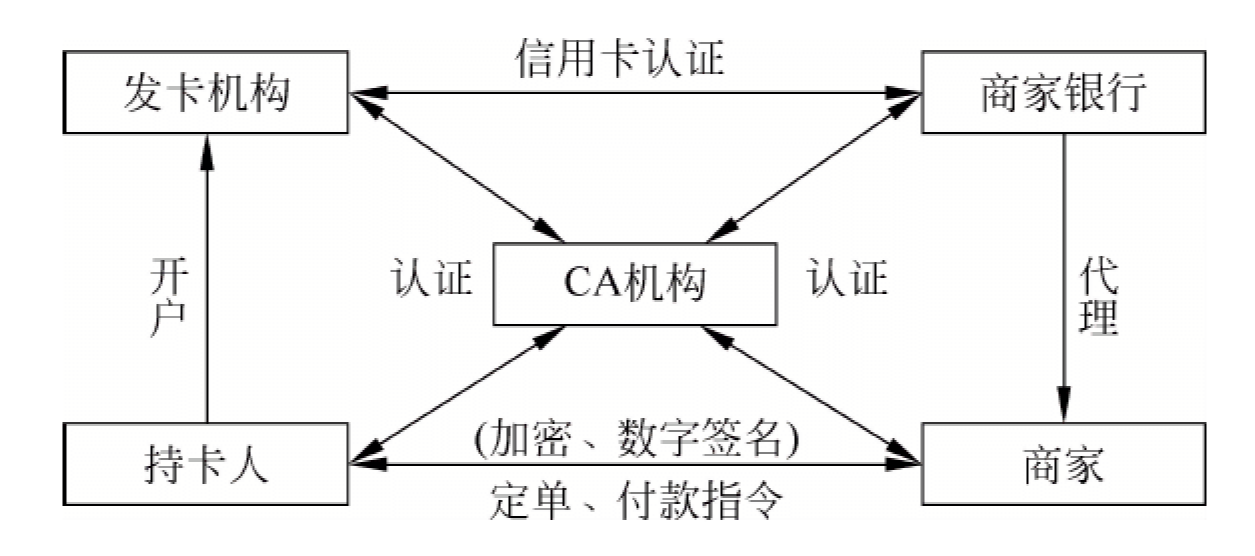


图 9.47　SET 协议示意图

在持卡交易中,持卡人、银行、商家之间存在以下的商业安全需求:

(1) 提供付款和订购的保密性。持卡人在商家订购商品的信息可能被第三方恶意利用,而且持卡人的账户信息也不希望被商家知道,因此需要提供付款和订购的保密性。

(2) 为持卡人的合法性提供认证。对持卡人的合法性通过证书和数字签名进行认证,可以防止第三方假冒持卡人的名义从商家订购商品。

(3) 为商家的合法性提供认证。持卡人需要识别为他们提供商品的商家是否合法,或许非法商家只是希望骗取持卡人的卡号和密码。对商家合法性的认证也是主要通过数字签

名和证书完成。

(4) 安全交易不依赖于安全传输机制。虽然 IPSec 和 SSL 的使用有助于保证交易安全和防止攻击者的入侵,但是仍然希望建立的 SET 协议能够提供在非安全通道上进行安全交易的机制。毕竟,大多数情况下,电子交易是通过 Internet 进行的。

此外,其他的商业安全需求还包括数据的完整性、不可抵赖性等。

完整的 SET 协议涉及以下几方面内容:

(1) 加密算法;

(2) 证书信息及格式;

(3) 购买信息及格式;

(4) 认可信息及格式;

(5) 划账信息及格式;

(6) 实体之间消息的传输协议。

下面对 SET 协议下的工作流程进行一个简单的描述。

(1) 持卡人开户。持卡人在一个支持电子支付的银行(如 MasterCard 或 Visa)开立信用卡账户,获得信用卡。

(2) 持卡人获得证书。持卡人收到来自银行签发的数字证书,证书中给定了持卡人的 RSA 公钥和有效期限。

(3) 商家获得证书。接受电子支付的商家必须拥有两种公钥证书,一种用于签名消息;另一种用于密钥交换。

(4) 持卡人订购商品。持卡人在商家的 Web 页面上查看商品目录,选择所需商品。当持卡人订购商品时,向商家发送订购请求,商家发回一个带有商品号、单价、总价、订购号的订购单。

(5) 持卡人对商家进行认证。商家在向持卡人发送订购单的同时,还发给持卡人一份商家的证书,持卡人通过证书验证正在进行交易的商家是否合法。

(6) 持卡人向商家发送订购和付款信息。持卡人将订购和支付信息以及持卡人的证书发送给商家。订购信息中确认了订购单中要购买商品的数量,用于商家组织送货。但是,支付信息中的信用卡细节,商家无法获得。商家也可以通过持卡人证书来验证持卡人的身份是否合法。

(7) 商家请求付款认证。商家向商家银行提出付款认证申请,希望商家银行验证持卡人信用卡是否合法以及信用卡的信用额度或余额是否足以支付这些商品。

(8) 商家确认订购。商家确认订购关系,并按持卡人指定的送货信息将商品发送给客户。客户可以是持卡人,也可以是持卡人指定的其他收货人。

(9) 商家请求付款。商家向商家银行请求支付货款,商家银行负责处理后续的与发卡银行之间的账务关系。

可以看到,SET 协议下的电子交易流程与实际购物流程非常接近,只不过所有交易是在互联网上进行。

9.5.2 数字信封和双重签名

在 SET 的交易过程中,数字信封和双重签名是保障交易安全的重要手段。数字信封的

技术前面已经介绍过，下面主要介绍双重签名技术的原理。

持卡人向商户提交订购信息的同时，也给银行提交付款信息，但有两个安全要求：

(1) 持卡人不希望商户知道自己的账户信息。

(2) 持卡人不希望开户行知道自己的消费内容。

要解决这个问题，需要引入双重签名技术。

在 SET 交易中，用 C(Customer)表示持卡人，M(Merchant)表示商家，B(Bank)表示银行。

如图 9.48 所示，双重签名的过程如下：

(1) C 产生发往 M 的订购信息 OI 和发往 B 的支付指令 PI；

(2) 对 OI 和 PI 做数字摘要，分别得到 H(OI)和 H(PI)，数字摘要的作用是防止消息在中途被篡改；

(3) 连接 H(OI)和 H(PI)得到 PO；

(4) 再对 PO 做数字摘要 H(PO)；

(5) 用 C 的私钥签名 H(PO)，得到 Sig(H(PO))，称为双重签名；

(6) C 将消息{OI，H(PI)，Sig(H(PO))}发给 M，将{PI，H(OI)，Sig (H(PO))}发给 B。

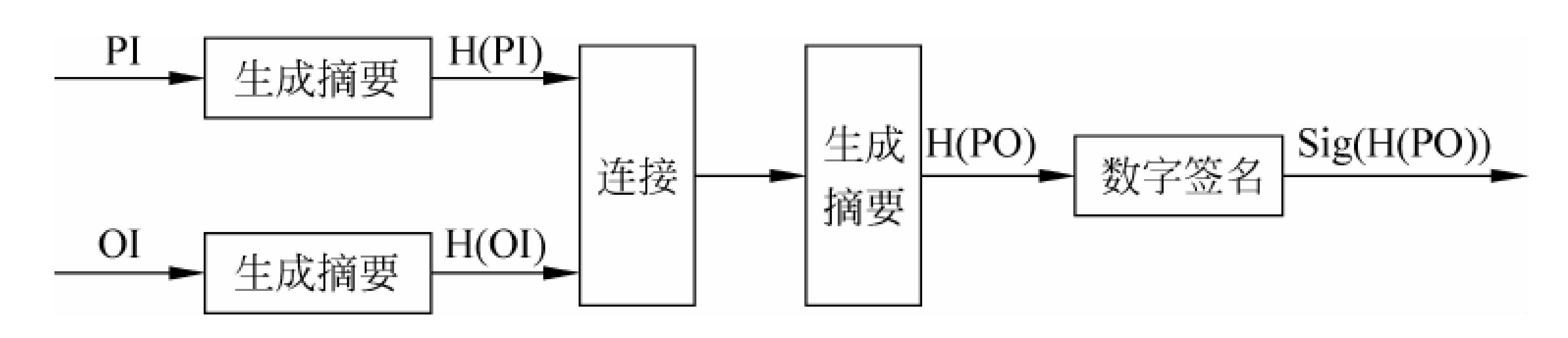


图 9.48　双重签名的生成过程

双重签名的验证过程如图 9.49 所示：

(1) M 接收到{OI，H(PI)，Sig(H(PO))}后，通过做信息摘要生成 H(OI)；

(2) B 接收到{PI，H(OI)，Sig (H(PO))}后，通过做信息摘要生成 H(PI)；

(3) M 将 H(OI)与 H(PI)连接并产生摘要 H(PO)；

(4) B 将 H(PI)与 H(OI)连接并产生摘要 H(PO)；

(5) M 验证 Sig(H(PO))是否为 H(PO)的签名；

(6) B 验证 Sig(H(PO))是否为 H(PO)的签名。

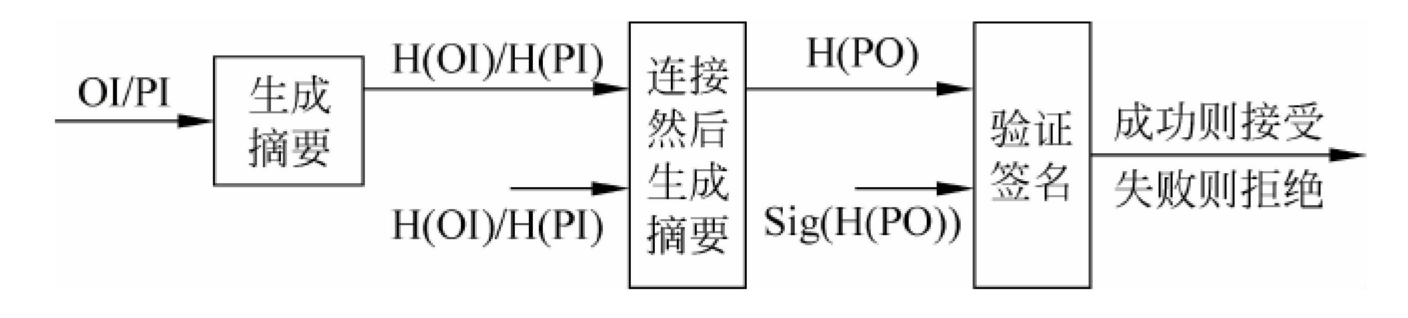


图 9.49　双重签名验证过程

9.6　习　　题

1. 简述 PPTP 与 L2TP 之间的异同点。

2. 什么是 CA？CA 有哪些作用？

3. 简述PPP协议的工作原理。

4. 在电子交易中,如果持卡人不希望商户知道自己的账号信息,且不希望开户行知道自己的消费内容,如何做到?

5. 与UNIX口令保护的相关的文件有________和________。

6. UNIX口令文件中的Salt值占________位,可转换成________个ASCII字符。

7. Windows系统的口令保护采用________或________算法。

8. PGP在加密前先要________,这样做节省了网络传输的时间和存储空间。

9. 由于许多电子邮件系统只允许使用ASCII码的正文,所以PGP提供了转换方案,将原始二进制流转化为________。

10. PGP信任模型中有两种信任:________和________。

11. ________是计算机网络安全中的一个重要技术,通过多种安全机制提供了链路层和网络层上的安全服务。

12. 根据建立VPN的目的可以将VPN分为________、________和________3类。

13. 在以太网上运行的PPP协议,称为________。

14. L2TP可以提供包头压缩。当压缩包头时,系统开销仅占用________个字节,而PPTP协议必须占用________个字节。

15. IKE协商分为________和________两个阶段。

16. IPSec协议主要由________协议、________协议两个协议组成。

17. IPSec有________和________两种工作模式。

18. SSL修改密码协议仅包含一条消息,该消息由一个值为________的字节构成。

19. ________是以信用卡支付为基础的网上电子支付系统规范。

第 10 章　密码学与安全编程

在编写密码学与信息安全的相关程序时，可选择多种语言，如 C++、Java、PHP 等。本章主要介绍基于 Java 平台的密码学与安全编程。

Java 的安全性一直是 Java 语言发展中的一个主要设计目标。经过不断地发展，目前已形成一整套的 Java 解决方案，包括 Java 加密体系结构（Java Cryptography Architecture，JCA）、Java 加密扩展（Java Cryptography Extension，JCE）、Java 认证与安全服务（Java Authentication And Authorization Service，JAAS）、Java 安全套接字扩展（Java Secure Socket Extension，JSSE）等。

其中，JCA 提供基本的加密解密框架，如证书、数字签名、消息摘要和密钥工厂。

JCE 在 JCA 的基础上做了扩展，提供各类加密、消息摘要和消息验证码等算法，如 DES、AES、RSA、DSA。

JSSE 提供基于 SSL 的安全通信功能。

JAAS 提供用户身份认证的功能。

结合前面章节讲过的内容，本章将介绍相关的编程方法与实现。

10.1　JCA

JCA 主要为 Java 应用提供基础安全服务。这些安全服务包括数字签名、消息摘要和密钥生成器。

JCA 的框架符合以下的设计理念：

（1）实现的独立性和互操作性；

（2）算法的独立性和可扩展性。

实现的独立性使 JCA 用户能轻松地运用密码学的概念进行编程，而不必关心这些概念的实现。

实现的独立性是通过“密码服务提供者”（Provider）的方式实现的。程序请求某类对象（如签名对象）实现某种服务（如 DSA 数字签名）时，可通过任一提供者获取具体实现方法。当有更高效或更安全的版本可用时，提供者的升级对应用程序来说完全是透明的。

实现的互操作性意味着对于相同的算法，由一个提供者生成的密钥可被另一个提供者使用，而一个提供者生成的签名也可由另一个提供者来验证。

算法独立性是通过定义密码“引擎”类来实现的，如 Message Digest 类、Signature 类和 Key Factory 类。

算法的可扩展性是指可以为 Engine 类添加新算法。

由于加密是一类非常重要的安全功能，美国曾经对加密技术的出口进行了限制，继而一度影响到 JCA 的功能。JCA 框架中并没有定义加密之类的机密性安全服务，而主要实现数

字签名、消息摘要等完整性安全服务。

10.1.1 消息摘要

java.security 包中的 MessageDigest 类提供了计算消息摘要的服务，具体步骤如下：

(1) 生成一个 MessageDigest 类，并确定 Hash 算法。

```
java.security.MessageDigest alga = java.security.MessageDigest.getInstance("SHA-1");
```

(2) 添加要进行计算摘要的消息。

```
alga.update(msgwithsign.getBytes());
```

(3) 计算出摘要。

```
byte[] digesta = alga.digest();
```

(4) 将消息和摘要发送给接收方。

(5) 接收方用相同的方法初始化，添加消息，然后比较摘要是否相同。

```
algb.isEqual(digesta,algb.digest());
```

MessageDigest 支持的算法包括 MD5、SHA-1、SHA-256 等。

下面给出完整的代码：

```
import java.security.*;
import javax.crypto.*;
public class TestSHA{
/*
 * @author Ray
 */
    public TestSHA() throws Exception{}
    public static void main(String[] args) throws Exception {
        String message1 = "TestOfMessageDigest";
        System.out.println("消息: " + message1);
        //生成一个消息摘要类,定义消息摘要算法
        java.security.MessageDigest md1 = java.security.MessageDigest.getInstance("SHA-384");
        //添加消息
        md1.update(message1.getBytes());
        //计算摘要
        byte[] digest1 = md1.digest();
        System.out.println("摘要: " + byte2hex(digest1));
        //接收方用相同的方法初始化,添加消息,然后比较摘要是否相同
        String message2 = "TestOfMessageDigest";
        java.security.MessageDigest md2 = java.security.MessageDigest.getInstance("SHA-384");
        md2.update(message2.getBytes());
        System.out.println("验证结果: " + md2.isEqual(digest1,md2.digest()));
    }
    //将 byte[]型转换成十六进制串
    public static String byte2hex(byte[] ba) {
        String strout = "";
        for (int i = 0;i<ba.length;i++){
```

```
            int j = ba[i] < 0?ba[i] + 256:ba[i];
            String str = Integer.toHexString(j);
            while (str.length()<2) str = '0' + str;
            if (i<ba.length - 1) strout + = (str + " - "); else strout + = str;
        }
        return strout.toUpperCase();
    }
}
```

由于消息摘要输出的是 byte 数组类型,不便输出打印,程序中增加了一个格式转换函数"byte2hex()",可将 byte 数组转换成十六进制串。

以上程序运行结果如下:

消息:TestOfMessageDigest

摘 要: ED-F7-11-D6-3E-B6-22-01-BF-A3-3E-FB-C2-57-80-BC-F5-45-A8-17-98-A0-15-24-EB-3F-91-4E-00-0B-6F-7F-E1-47-F8-57-47-70-FE-2B-58-F3-19-E4-0C-32-B1-B2

验证结果:true

本例中由于使用的消息摘要算法是 SHA-384,最终计算出的消息摘要是 384 位的数据。

10.1.2　密钥生成与数字签名

数字签名一般采用公钥体制,java.security 包中的 KeyPairGenerator 类提供了产生密钥对的方法。有了密钥对后,java.security 包中 Signature 类则提供了计算数字签名的方法。具体步骤如下:

(1) 生成密钥对。

对于用户来说,密钥对的生成非常重要,无论是数字签名还是加密解密都需要用到密钥对。密钥对生成后,应分别保存,私钥通常存储在本地,公钥则可提供给他人。

密钥对的生成通常使用 KeyPairGenerator 类,如:

```
java.security.KeyPairGenerator  keygen = java.security.KeyPairGenerator.getInstance("DSA");
```

可以使用随机参数产生器设置种子,并初始化参数:

```
SecureRandom secrand = new SecureRandom();
secrand.setSeed("xxxx".getBytes());
keygen.initialize(512,secrand);
```

初始化是可选的:

```
keygen.initialize(512);
```

初始化后就可以生成密钥对了,密钥对包括公钥和私钥:

```
KeyPair keys = keygen.generateKeyPair();
PublicKey pubkey = keys.getPublic();
PrivateKey prikey = keys.getPrivate();
```

通常,密钥需要通过文件的形式保存起来,以便将来再使用:

```
out = new java.io.ObjectOutputStream(new java.io.FileOutputStream("pubkey.dat"));
out.writeObject(pubkey);
```

(2) 数字签名需要用到私钥，可从私钥文件中读入私钥：

```
java.io.ObjectInputStream in = new java.io.ObjectInputStream(new java.io.FileInputStream
("prikey.dat"));
PrivateKey prikey = (PrivateKey)in.readObject();
```

签名时，先初始化一个 Signature 对象，然后用私钥对消息签名：

```
java.security.Signature sign1 = java.security.Signature.getInstance("DSA");
sign1.initSign(prikey);
sign1.update(msgwithsign.getBytes());
byte[] signofmsg = sign1.sign();
```

可以把消息和签名保存在同一个文件中：

```
java.io.ObjectOutputStream out = new java.io.ObjectOutputStream(new java.io.FileOutputStream
("msgwithsign.dat"));
out.writeObject(msgwithsign);
out.writeObject(signofmsg);
```

消息和摘要也可以分别存放在两个文件中保存或发送。

(3) 接收方收到发送方发来的消息和摘要，并通过某种方式得到发送方的公钥。接下来便可以用公钥对消息和签名进行验证。

首先从文件中重建公钥：

```
java.io.ObjectInputStream in = new java.io.ObjectInputStream(newjava.io.FileInputStream
("pubkey.dat"));
PublicKey pubkey = (PublicKey)in.readObject();
```

然后读入消息和签名：

```
in = new java.io.ObjectInputStream(new java.io.FileInputStream("msgwithsign.dat"));
String msg = (String)in.readObject();
byte[] signofmsg = (byte[])in.readObject();
```

接下来初始一个 Signature 对象，并用公钥对签名进行验证：

```
java.security.Signature sign2 = java.security.Signature.getInstance("DSA");
sign2.initVerify(pubkey);
sign2.update(msg.getBytes());
if (sign2.verify(signofmsg))System.out.println("签名有效");
```

以下给出数字签名的完整代码：

```
import java.security.*;
import java.security.spec.*;
/*
*     @author Ray
*/
public class TestDSA {
```

```
        public static void main(String[] args) throws
        java.security.NoSuchAlgorithmException,java.lang.Exception {
        TestDSA my = new TestDSA();
        my.run();
    }
    public void run(){
        //生成密钥对: 如果已经生成过,本过程就可以跳过
        //发送方的私钥文件 prikey.dat 要保存在本地,而公钥文件 pubkey.dat 则发送给接收方
        if ((new java.io.File("prikey.dat")).exists() == false) {
            if (generatekey() == false) {
                System.out.println("生成密钥对失败");
                return;
            }
        }
        // 发送方从文件中读入私钥,对消息进行签名后保存在一个文件(msgwithsign.dat)中
        //然后再把 msgwithsign.dat 发送给接收方,
        //数字签名可以放进 msgwithsign.dat 文件中,也可分别发送
        try {
            java.io.ObjectInputStream in = new java.io.ObjectInputStream(new
            java.io.FileInputStream("prikey.dat"));
            PrivateKey prikey = (PrivateKey)in.readObject();
            in.close();
            String msgwithsign = "TestOfDSA"; //要签名的信息
            //用私钥对信息生成数字签名
            java.security.Signature sign1 = java.security.Signature.getInstance("DSA");
            sign1.initSign(prikey);
            sign1.update(msgwithsign.getBytes());
            byte[] signofmsg = sign1.sign(); //对信息的数字签名
            System.out.println("数字签名: " + byte2hex(signofmsg));
            //此处把信息和数字签名保存在一个文件中
            java.io.ObjectOutputStream out = new java.io.ObjectOutputStream(new
            java.io.FileOutputStream("msgwithsign.dat"));
            out.writeObject(msgwithsign);
            out.writeObject(signofmsg);
            out.close();
            System.out.println("签名并生成文件成功");
        } catch (java.lang.Exception e) {
            e.printStackTrace();
            System.out.println("签名时发生错误!");
        }
        //接收方通过某种方式得到发送方的公钥和签名文件,
        //然后用公钥对签名进行验证
        try {
            java.io.ObjectInputStream in = new java.io.ObjectInputStream(new
            java.io.FileInputStream("pubkey.dat"));
            PublicKey pubkey = (PublicKey)in.readObject();
            in.close();
            in = new java.io.ObjectInputStream(new
            java.io.FileInputStream("msgwithsign.dat"));
            String msg = (String)in.readObject();
            byte[] signofmsg = (byte[])in.readObject();
```

```
                in.close();
                java.security.Signature sign2 = java.security.Signature.getInstance("DSA");
                sign2.initVerify(pubkey);
                sign2.update(msg.getBytes());
                if (sign2.verify(signofmsg)) {
                    System.out.println("消息内容: " + msg);
                    System.out.println("签名有效");
                } else
                    System.out.println("非正常签名");
            } catch (java.lang.Exception e) {e.printStackTrace();};
        }
        //生成密钥对的函数
        public boolean generatekey() {
            try {
                java.security.KeyPairGenerator
                keygen = java.security.KeyPairGenerator.getInstance("DSA");
                keygen.initialize(512);
                KeyPair keys = keygen.genKeyPair();
                PublicKey pubkey = keys.getPublic();
                PrivateKey prikey = keys.getPrivate();
                java.io.ObjectOutputStream out = new java.io.ObjectOutputStream(new
                java.io.FileOutputStream("prikey.dat"));
                out.writeObject(prikey);
                out.close();
                System.out.println("写入对象 prikeys ok");
                out = new java.io.ObjectOutputStream(new
                java.io.FileOutputStream("pubkey.dat"));
                out.writeObject(pubkey);
                out.close();
                System.out.println("写入对象 pubkeys ok");
                System.out.println("生成密钥对成功");
                return true;
            }   catch (java.lang.Exception e) {
                e.printStackTrace();
                System.out.println("生成密钥对失败");
                return false;
            }
        }
    }
```

函数 byte2hex()与前面的例子完全相同，在此省去。

运行结果如下：

数字签名：30-2D-02-15-00-8F-CD-CA-39-76-CB-85-99-F8-77-DB-6B-65-71-D2-E6-2D-AF-AF-59-02-14-32-4D-3E-D8-38-F8-58-BD-F6-DC-D4-0E-9A-CF-21-CB-6C-99-B8-B3

签名并生成文件成功

消息内容：TestOfDSA

签名有效。

在以上实例中，密钥的保存是通过对象流的方式保存和传送的，也可以采用相关的标准。

公钥通常采用 X.509 标准进行编码,私钥则一般采用 PKCS＃8 编码。

使用 X.509 标准的公钥编码和重建,可参考以下代码:

```
byte[ ] encodedPubKey = pubkey.getEncoded();
X509EncodedKeySpecpubKeySpec = new X509EncodedKeySpec(encodedPubKey);
KeyFactory keyFactory = KeyFactory.getInstance("DSA");
PublicKeypubKey = keyFactory.generatePublic(pubKeySpec);
```

使用 PKCS＃8 进行私钥编码和重建的代码,可参考以下代码:

```
byte[ ]encodedPKCS8 = prikey.getEncoded();
PKCS8EncodedKeySpec priPKCS8 = new PKCS8EncodedKeySpec(encodedPKCS8);
KeyFactory keyFactory = KeyFactory.getInstance("DSA");
PrivateKey prikey = keyFactory.generatePrivate(priPKCS8);
```

10.1.3　数字证书

Java 中提供了一个建立和管理密钥库及数字证书的工具,即 keytool 工具。

keytool 工具将密钥库实现为一个文件,并通过口令来保护。密钥库中可存放两类条目,一类条目是私钥,另一类条目是可信任的证书。私钥受口令保护,公钥不受口令保护。

下面先用 keytool 工具创建一个密钥库。

创建密钥库需使用-genkey 选项,默认状态下,创建者需要登记姓名、组织单位、组织名称、城市或区域名称等信息。密钥库的口令至少是 6 个字符。

创建密钥库时,至少需要存储一份密钥,默认状态下是 mykey 用户的密钥。

mykey 的口令可以不填,此时口令与密钥库的口令一致。创建密钥库的过程如图 10.1 所示。

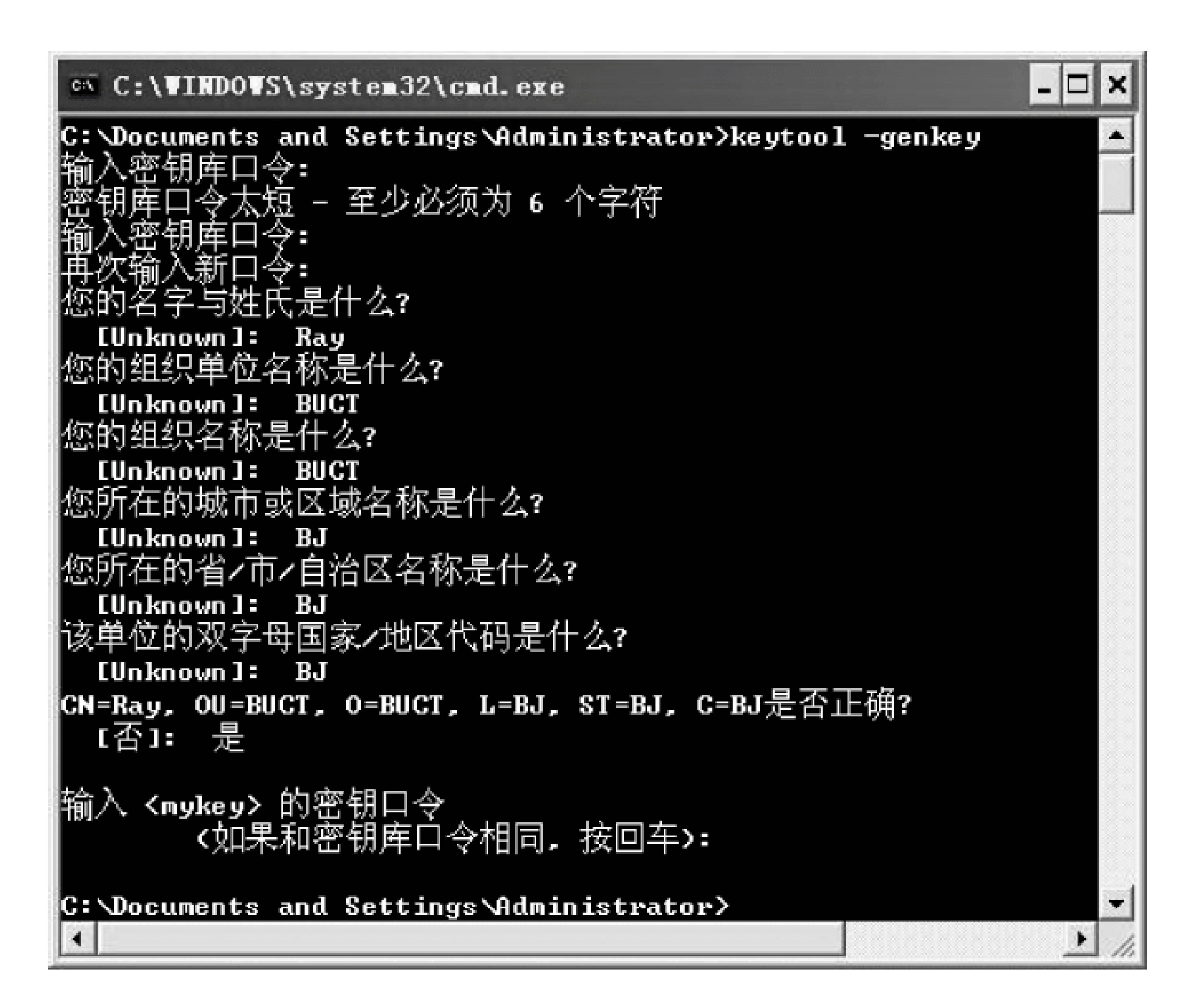


图 10.1　密钥库的创建

使用-list 参数可以列出密钥库中存储的所有条目,如图 10.2 所示。

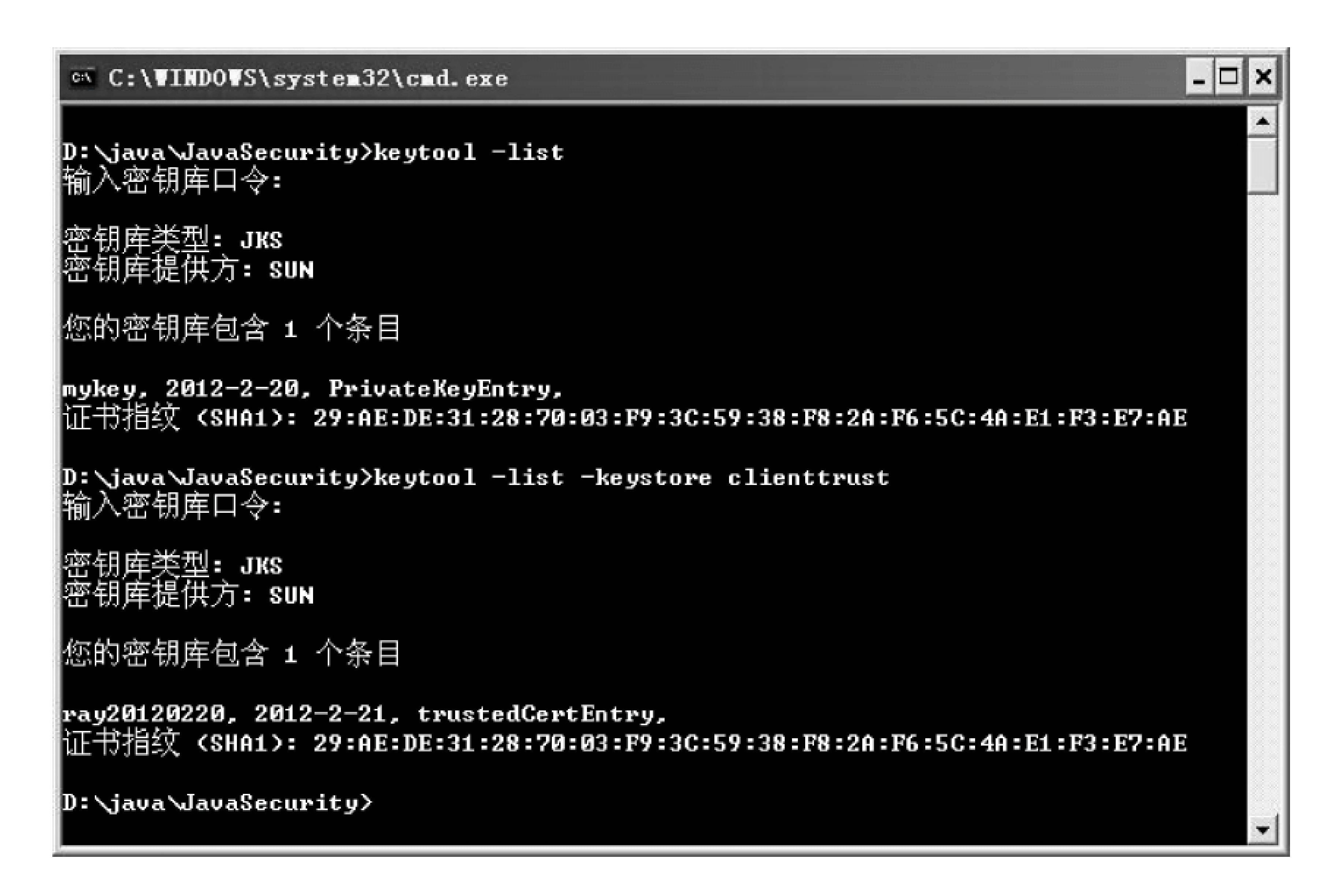


图 10.2　密钥库中的条目显示

通过-keystore 参数还可以指定具体的密钥库的名称。从图 10.2 中可以看到 mykey 的条目类型是 PrivateKeyEntry，表示是私钥条目，而 ray20120220 的条目类型是 trustedCertEntry，表示信任的证书条目。

对于 mykey 这类私钥条目，还可以导出对应的证书，如图 10.3 所示。

导出证书使用-export 选项，并通过-alias 指定条目的名称，通过-file 指定导出证书文件的文件名。



图 10.3　证书的导出

keytool 还能够显示证书文件内容的功能，如图 10.4 所示。具体语句是：

```
keytool - printcert  - file ray20120220.cer
```

keytool 工具固然方便，但如果要在程序中实现对密钥库和数字证书的访问，则需要用到 java.security.cert 包。

接下来的程序利用 java.security.cert 包，通过编程的方式来读取证书的详细条目。

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.math.*;
```

```
public class TestCerReadItem{
/*
 *     @author Ray
 */
public static void main(String args[ ]) throws Exception{
      CertificateFactory certfac = CertificateFactory.getInstance("X.509");
      FileInputStream in = new FileInputStream("ray20120220.cer");
      java.security.cert.Certificate cert = certfac.generateCertificate(in);
      in.close();
      X509Certificate x509cert = (X509Certificate) cert;
      System.out.println("版本号: " + x509cert.getVersion());
      System.out.println("序列号: " + x509cert.getSerialNumber().toString(16));
      System.out.println("主题: " + x509cert.getSubjectDN());
      System.out.println("签发者: " + x509cert.getIssuerDN());
      System.out.println("有效期起始日: " + x509cert.getNotBefore());
      System.out.println("有效期失效日: " + x509cert.getNotAfter());
      System.out.println("签名算法: " + x509cert.getSigAlgName());
      byte[ ] sign = x509cert.getSignature();
      System.out.println("签名: " + new BigInteger(sign).toString(16));
      PublicKey pubkey = x509cert.getPublicKey();
      byte[ ] pubkeyenc = pubkey.getEncoded();
          System.out.println("公钥: " + byte2hex(pubkeyenc));
      }
}
```

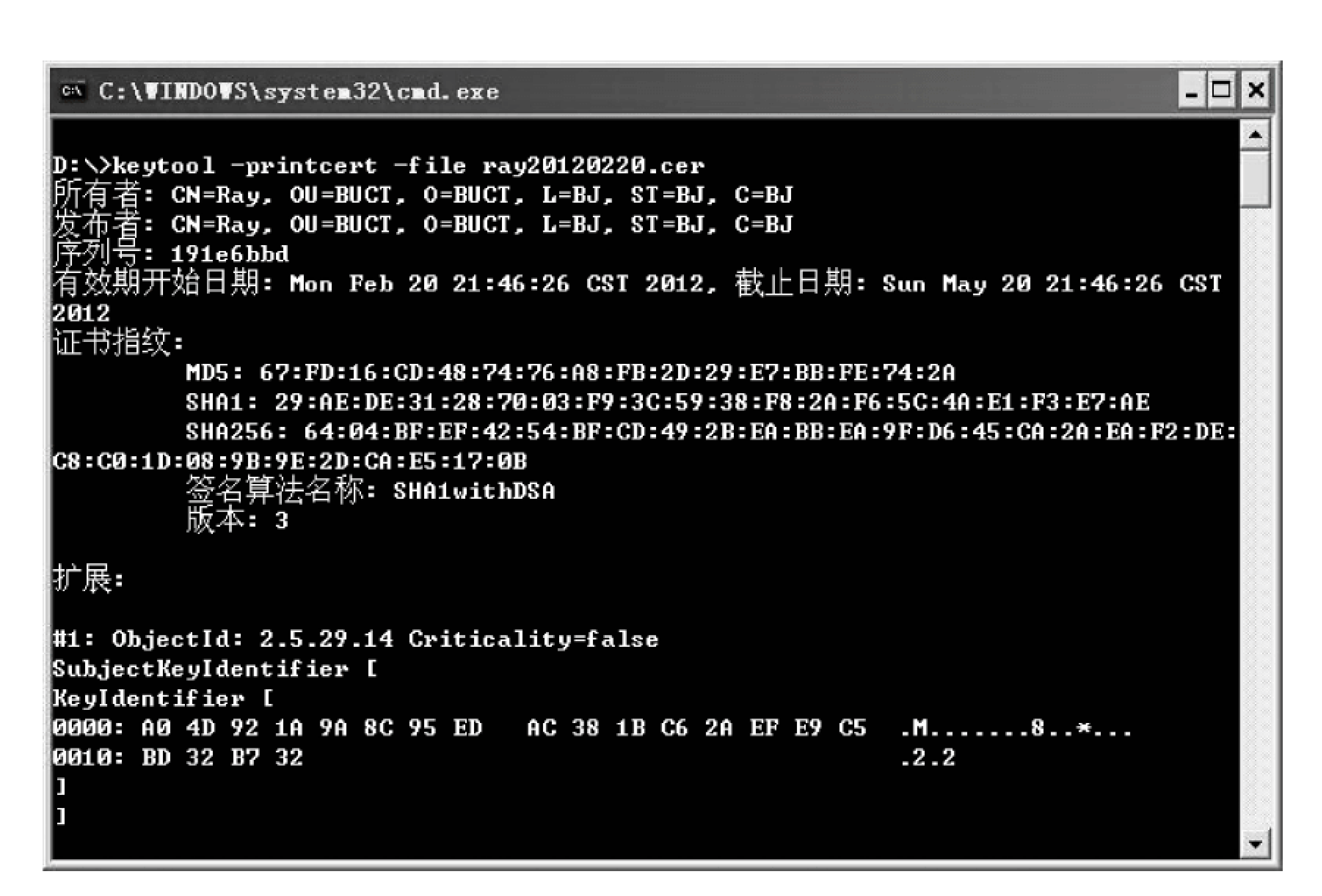


图 10.4　显示证书内容

函数 byte2hex()与前面的例子完全相同,在此省去。

运行结果如下:

版本号:3

序列号:191e6bbd

全名：CN=Ray，OU=BUCT，O=BUCT，L=BJ，ST=BJ，C=BJ

签发者全名：CN=Ray，OU=BUCT，O=BUCT，L=BJ，ST=BJ，C=BJ

有效期起始日：Mon Feb 20 21:46:26 CST 2012

有效期截止日：Sun May 20 21:46:26 CST 2012

签名算法：SHA-1withDSA

签名：302c02145e2f79785381fee3792ca83573bd4abdbd931e1e02142ea09dcaca0dfe7618b66874
51698005e709b4fd

公钥：308201B83082012C06072A8648CE3804013082011F02818100FD7F53811D75122952D
F4A9C2EECE4E7F611B7523CEF4400C31E3F80B6512669455D402251FB593D8D58FABFC5
F5BA30F6CB9B556CD7813B801D346FF26660B76B9950A5A49F9FE8047B1022C24FBBA9
D7FEB7C61BF83B57E7C6A8A6150F04FB83F6D3C51EC3023554135A169132F675F3AE2B6
1D72AEFF22203199DD14801C70215009760508F15230BCCB292B982A2EB840BF0581CF50
2818100F7E1A085D69B3DDECBBCAB5C36B857B97994AFBBFA3AEA82F9574C0B3D0782
675159578EBAD4594FE67107108180B449167123E84C281613B7CF09328CC8A6E13C167A8
B547C8D28E0A3AE1E2BB3A675916EA37F0BFA213562F1FB627A01243BCCA4F1BEA8519
089A883DFE15AE59F06928B665E807B552564014C3BFECF492A0381850002818100B905D8
4204CEFC57440B464FCC3AAC46AF403D4369EE5B8105CE6E0FD026887D77C89B51A73D
EA96065232C0D5B946B3316AAEF64EE47A47F91644024589C9DFDD095ED10CC4C259280
A2B91A26D972B70B9A1F6AB21015AA18E727A28E69C553444A6BC9346A177B678DE4B3
0A8EAF00E25745841BBD7A9BFA275DD83FA662C

10.2 JCE

为了提供对机密性安全服务的支持，Java 平台提供了一种类似于 JCA 框架思想的 JCE 框架。

在 Java 2 SDK 的 1.2.x 及 1.3.x 版本中，JCE 都是以可选包的形式提供的。但从 1.4 版本后，由于美国对加密技术出口限制的放宽，JCE 也被集成到了 Java 的 SDK 中，直到今天。

JCE 提供了对于加密解密、密钥管理、消息验证码等算法的实现。

10.2.1 对称密码算法

JCE 对对称密码算法的支持非常强大，可支持的加密算法包括 Blowfish、DES、DESede 和 AES 等。

对称密码算法加密主要用到以下几个类。

(1) Cipher 类：包括密码选择、加密与解密。它是 JCE 框架中的核心部分。

(2) KeyGenerator 类：用来产生各类算法的密钥，通过 getInstance 方法建立，并用 init 方法初始化。

(3) SecretKeyFactory 类：主要处理对称算法的密钥产生与转换。

使用对称密码算法进行加密解密的一般步骤如下：

(1) 首先生成密钥并保存。

```
KeyGenerator keygen = KeyGenerator.getInstance(Algorithm);
SecretKey aeskey = keygen.generateKey();
```

(2) 用密钥加密明文，生成密文。

```
Cipher c1 = Cipher.getInstance(Algorithm);
c1.init(Cipher.ENCRYPT_MODE,aeskey);
byte[] cipherByte = c1.doFinal(plaintext.getBytes());
```

(3) 接收方收到密文和密钥，然后用密钥解密密文。密文的发送可以在非安全通道上进行，但密钥的发送必须通过安全通道。

```
c1 = Cipher.getInstance(Algorithm);
c1.init(Cipher.DECRYPT_MODE,aeskey);
byte[] decryptoByte = c1.doFinal(cipherByte);
```

密钥 aeskey 可以编码成"密钥材料"后，存入密钥文件中：

```
java.io.ObjectOutputStream
out = new java.io.ObjectOutputStream(new java.io.FileOutputStream("aeskey.dat"));
out.writeObject(aeskey);
out.close();
```

当接收方使用该密钥时，可从密钥文件中读出"密钥材料"，然后重建密钥：

```
java.io.ObjectInputStream in = new java.io.ObjectInputStream(new java.io.FileInput
Stream("aeskey.dat"));
Key aeskey2 = (Key)in.readObject();
in.close();
```

以下是完整的加密算法的源代码：

```
import java.security.*;
import javax.crypto.*;
public class TestAES {
/*
 *     @author Ray
 */
    public static void main(String[] args){
        TestAES my = new TestAES();
        my.run();
    }
    public void run() {
        Security.addProvider(new com.sun.crypto.provider.SunJCE());
        String Algorithm = "AES";
        String plaintext = "TestOfEncryption";
        try {
        //生成密钥
        KeyGenerator keygen = KeyGenerator.getInstance(Algorithm);
        SecretKey aeskey = keygen.generateKey();
```

```
            //密钥材料存储于本地
            java.io.ObjectOutputStream out = new java.io.ObjectOutputStream(new
            java.io.FileOutputStream("aeskey.dat"));
            out.writeObject(aeskey);
            out.close();
            //加密
            System.out.println(plaintext);
            Cipher c1  =  Cipher.getInstance(Algorithm);
            c1.init(Cipher.ENCRYPT_MODE,aeskey);
            byte[] cipherByte = c1.doFinal(plaintext.getBytes());
            System.out.println(byte2hex(cipherByte));
            //重建密钥
            java.io.ObjectInputStream in = new java.io.ObjectInputStream(new
            java.io.FileInputStream("aeskey.dat"));
            Key aeskey2 = (Key)in.readObject();
            in.close();
            //解密
            c1  =  Cipher.getInstance(Algorithm);
            c1.init(Cipher.DECRYPT_MODE,aeskey2);
            byte[] decryptoByte = c1.doFinal(cipherByte);
            System.out.println(byte2hex(decryptoByte));
            System.out.println(new String(decryptoByte));
        }
            catch (java.security.NoSuchAlgorithmException e1) {e1.printStackTrace();}
            catch (javax.crypto.NoSuchPaddingException e2) {e2.printStackTrace();}
            catch (java.lang.Exception e3) {e3.printStackTrace();}
        }
    }
```

函数 byte2hex()与前面的例子完全相同,在此省去。

运行结果如下:

TestOfEncryption

61-65-04-02-65-E8-4B-51-07-19-08-B3-C3-6B-F3-B1-30-42-38-56-42-87-77-36-6F-13-84-E3-39-BD-D6-41

54-65-73-74-4F-66-45-6E-63-72-79-70-74-69-6F-6E

TestOfEncryption

以上例子中的加密算法采用的是 AES 算法。默认情况下,加密算法采用的加密模式是 ECB 模式。

10.2.2 公钥密码算法

Java 从 1.4 版本开始支持公钥密码算法中的 RSA 算法,之后又陆续支持一些其他的公钥密码算法。例如,Java 7.0 版本已开始支持 ECC 算法。

与数字签名不同,公钥密码算法中用公钥进行加密,用私钥进行解密,具体步骤如下:

(1) 生成密钥对。密钥对生成后,应分别保存,私钥通常存储在本地,公钥则可提供给消息发送方。

密钥对的生成仍使用 KeyPairGenerator 类:

```
KeyPairGenerator keygen = KeyPairGenerator.getInstance("RSA");
keygen.initialize(1024);
KeyPair keys = keygen.genKeyPair();
PublicKey pubkey = keys.getPublic();
PrivateKey prikey = keys.getPrivate();
```

密钥一般以文件的形式存储：

```
ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream ("RSApri key.dat"));
out.writeObject(prikey);
out = new ObjectOutputStream(new FileOutputStream("RSApubkey.dat"));
out.writeObject(pubkey);
```

(2) 加密需要用到公钥，可从公钥文件中获取公钥参数：

```
ObjectInputStream in = new ObjectInputStream(new FileInputStream("RSApub key.dat"));
RSAPublicKey pubkey = (RSAPublicKey)in.readObject();
BigInteger e = pubkey.getPublicExponent();
BigInteger n = pubkey.getModulus();
```

(3) 加密是一个模指数运算：

```
BigInteger cipherBI = plainBI.modPow(e,n);
```

(4) 可把密文保存到本地文件中：

```
String ciphertext = cipherBI.toString( );
BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutput
Stream("RSACipher.dat")));
out.write(ciphertext,0,ciphertext.length( ));
```

(5) 解密需要用到私钥，可从私钥文件中获取私钥参数。

```
ObjectInputStream in = new ObjectInputStream(new FileInputStream("RSAprikey.dat"));
RSAPrivateKey prikey = (RSAPrivateKey)in.readObject();
BigInteger d = prikey.getPrivateExponent();
BigInteger n = prikey.getModulus();
```

(6) 解密也是一个模指数运算。

```
BigInteger plainBI = cihperBI.modPow(d,n);
```

以下是实现 RSA 密码算法的代码：

```
import java.security.*;
import java.security.spec.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import javax.crypto.interfaces.*;
import java.security.interfaces.*;
import java.math.*;
import java.io.*;
/*
 *    @author Ray
```

```
  */
public class TestRSA {
        public static void main(String[] args) throws
        java.security.NoSuchAlgorithmException,java.lang.Exception {
        TestRSA my = new TestRSA();
        my.run();
    }
    public void run(){
        //生成密钥对:如果已经生成过,本过程就可以跳过
        if ((new File("RSAprikey.dat")).exists() = = false) {
            if (generatekey() = = false) {
                System.out.println("生成密钥对失败");
                return;
            }
        }
        try {
        //读入公钥文件,重建公钥
        ObjectInputStream in = new ObjectInputStream(new FileInputStream("RSApubkey. dat"));
        RSAPublicKey pubkey = (RSAPublicKey)in.readObject();
        in.close();
        //从公钥中读出参数 e 和 n
        BigInteger e = pubkey.getPublicExponent();
        BigInteger n = pubkey.getModulus();
        System.out.println("公钥参数 e: " + e);
        System.out.println("公钥参数 n: " + n);
        // 给定明文编码成大整数
        String plaintext = new String("TestOfRSA");
        byte plainbytes[] = plaintext.getBytes("UTF8");
        BigInteger plainBI = new BigInteger(plainbytes);
        System.out.println("明文: " + plaintext);
        System.out.println("明文编码: " + plainBI.toString());
        // 计算密文
        BigInteger cipherBI = plainBI.modPow(e,n);
        System.out.println("密文编码: " + cipherBI.toString());
        // 把密文保存到本地文件中
        String ciphertext = cipherBI.toString( );
        BufferedWriter out =  new BufferedWriter(new OutputStreamWriter(new
        FileOutputStream("RSACipher.dat")));
        out.write(ciphertext,0,ciphertext.length( ));
        out.close( );
        } catch (java.lang.Exception e) {
            e.printStackTrace();
            System.out.println("加密时发生错误!");
        }
        try {
        //通过私钥文件重建私钥
        ObjectInputStream in = new ObjectInputStream(new FileInputStream("RSAprikey.dat"));
        RSAPrivateKey prikey = (RSAPrivateKey)in.readObject();
        in.close();
        //获取私钥参数
        BigInteger d = prikey.getPrivateExponent();
```

```
            BigInteger n = prikey.getModulus();
            System.out.println("私钥参数 d: " + d);
            System.out.println("私钥参数 n: " + n);
            //读出密文
            BufferedReader incipher = new BufferedReader(new InputStreamReader(new
            FileInputStream("RSACipher.dat")));
            String ciphertext = incipher.readLine();
            incipher.close();
            BigInteger cihperBI = new BigInteger(ciphertext);
            //解密
            BigInteger plainBI = cihperBI.modPow(d,n);
            System.out.println("解出明文的编码: " + plainBI);
            byte[] plainbytes = plainBI.toByteArray();
            System.out.print("解出的明文: ");
            for(int i = 0;i < plainbytes.length;i++) {System.out.print((char) plainbytes[i]);}
            } catch (java.lang.Exception e) {e.printStackTrace();};
        }
        //生成密钥对的函数
        public boolean generatekey() {
            try {
            java.security.KeyPairGenerator
            keygen = java.security.KeyPairGenerator.getInstance("RSA");
            keygen.initialize(1024);
            KeyPair keys = keygen.genKeyPair();
            PublicKey pubkey = keys.getPublic();
            PrivateKey prikey = keys.getPrivate();
            ObjectOutputStream out = new ObjectOutputStream
            (new FileOutputStream("RSAprikey. dat"));
            out.writeObject(prikey);
            out.close();
            System.out.println("写入对象 prikeys ok");
            out = new ObjectOutputStream(new FileOutputStream("RSApubkey.dat"));
            out.writeObject(pubkey);
            out.close();
            System.out.println("写入对象 pubkeys ok");
            System.out.println("生成密钥对成功");
            return true;
            } catch (java.lang.Exception e) {
            e.printStackTrace();
            System.out.println("生成密钥对失败");
            return false;
            }
        }
    }
```

程序运行结果如下：

公钥参数 e：65537

公钥参数 n：1029853236451124623299227409355597160619204273748287730729242572771496568381917596997544473198386090187031828262013342235803122278873478921245568985416941030099883316750865630909233453691145662719424795569576107647950336866 1

8238920649707841805510853582976550530973662225287933849396152214489839172755081
971367

明文：TestOfRSA

明文编码：1556836816696122692417

密文编码：3018667338167055217303435975006809459590420240079233840867179625771470563844767605877480604677519096285959189871813129196010700430874342534948903190697403367525203965991925538850443623168049941854414621402416807678618796423744733750080417545703449410466051963818443101995650742001160779422454162159404516290 2

私钥参数 d：5587610263168758007732109834424267793353016702922259964488824237088800979374128494444159234934191768752075583157707924259683449348029840163823480168981732713615949283609015325191166656292543837507105691861233964073575749764373629943147196998429617025697263385269072093852066841328242577209529509652718579915 3

私钥参数 n：1029853236451124623299227409355597160619204273748287730729242572771496568381917596997544473198386090187031828262013342235803122278873478921245568985416941030099883316750865630909233453691145662719424795569576107647950336866182389206497078418055108535829765505309736622252879338493961522144898391727550819 71367

解出明文的编码：1556836816696122692417

解出的明文：TestOfRSA

程序中密码算法的参数往往非常大，所以在运算中必须使用 BigInteger 类。

10.2.3 消息验证码

消息验证码的核心类是 Mac 类。

计算消息验证码的步骤：先通过 getInstance 建立 Mac 对象，再用 init 方法进行初始化，最后用 doFinal()方法来计算 MAC 值。

以下给出计算 MAC 的源程序：

```
import java.io.*;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
public class TestMAC{
/*
*     @author Ray
*/
public static void main(String args[]) throws Exception{
//设置密钥
    String key = "helloworld";
    System.out.println("密钥: " + key);
    SecretKeySpec sks = new SecretKeySpec(key.getBytes(),"HmacMD5");
    //生成 MAC 实例并初始化
```

```
        Mac mac1 = Mac.getInstance("HmacMD5");
        mac1.init(sks);
        String msg = "TestofMAC";
        System.out.println("消息: " + msg);
        mac1.update(msg.getBytes());
        System.out.println("MAC: " + byte2hex(mac1.doFinal()));
        }
}
```

函数 byte2hex()与前面的例子完全相同,在此省去。

运行结果如下:

密钥:helloworld

消息:TestOfMAC

MAC:EC-C6-8B-2B-E7-97-B5-A9-06-70-EE-95-31-41-86-CE

本例中采用的是 HmacMD5 算法,此外 Java 还支持 HmacSHA-1 等算法。

10.2.4 Diffie-Hellman 密钥协商协议

前面已介绍过 Diffie-Hellman 密钥协商协议的基本原理,密钥协商主要用在不安全通道中,通信双方协商出一个共享的会话密钥。

这里不妨假设通信双方分别是 Alice 和 Bob,JCE 中实现的 Diffie-Hellman 密钥协商协议可通过以下步骤实现:

(1) 如果 Alice 是通信发起方,Alice 先生成 DH 类型的密钥对。密钥对的产生需要一定的时间,所以密钥对生成后可保存下来供下次继续使用。

密钥对的产生仍使用 KeyPairGenerator 类:

```
KeyPairGenerator AliceKpairGen = KeyPairGenerator.getInstance("DH");
AliceKpairGen.initialize(1024);
KeyPair AliceKpair = AliceKpairGen.generateKeyPair();
```

(2) Alice 将密钥对中的公钥发送给 Bob。Bob 从 Alice 发送来的公钥中读出 DH 密钥对的初始化参数,用它生成 Bob 的 DH 密钥对。

需要注意的是 Bob 与 Alice 具有相同的初始化参数,是 DH 密钥协商过程实现的前提。

```
byte[] AlicePubKeyEnc = AliceKpair.getPublic().getEncoded();
KeyFactory BobKeyFac = KeyFactory.getInstance("DH");
X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec
(AlicePubKeyEnc);
PublicKey AlicePubKey = BobKeyFac.generatePublic(x509KeySpec);
DHParameterSpec dhParamSpec = ((DHPublicKey)AlicePubKey).getParams();
KeyPairGenerator BobKpairGen = KeyPairGenerator.getInstance("DH");
BobKpairGen.initialize(dhParamSpec);
KeyPair BobKpair = BobKpairGen.generateKeyPair();
```

Bob 凭借自己的私钥和 Alice 的公钥生成用于 DES 加密的会话密钥。

```
KeyAgreement BobKeyAgree = KeyAgreement.getInstance("DH");
BobKeyAgree.init(BobKpair.getPrivate());
```

```
BobKeyAgree.doPhase(AlicePubKey, true);
SecretKey BobSessionKey = BobKeyAgree.generateSecret("DES");
```

虽然 Bob 已拥有了会话密钥，但 Alice 还不知道这个会话密钥。为此，Bob 将他的 DH 密钥对中的公钥发送给 Alice，这样 Alice 就可以根据 Bob 的公钥来生成一个相同的会话密钥了。

```
byte[] BobPubKeyEnc = BobKpair.getPublic().getEncoded();
KeyFactory AliceKeyFac = KeyFactory.getInstance("DH");
x509KeySpec = new X509EncodedKeySpec(BobPubKeyEnc);
PublicKey BobPubKey = AliceKeyFac.generatePublic(x509KeySpec);
KeyAgreement AliceKeyAgree = KeyAgreement.getInstance("DH");
AliceKeyAgree.init(AliceKpair.getPrivate());
AliceKeyAgree.doPhase(BobPubKey, true);
SecretKey AliceSessionKey = AliceKeyAgree.generateSecret("DES");
```

有了相同的会话密钥，现在 Alice 和 Bob 就可以借助会话密钥进行 DES 算法的加密解密了。

以下是完整的源代码：

```
import java.security.*;
import java.security.spec.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import javax.crypto.interfaces.*;
public class TestDHKey {
/*
*     @author Ray
*/
public static void main(String argv[]) {
    try {
        TestDHKey my = new TestDHKey();
        my.run();
        } catch (Exception e) {     System.err.println(e);     }
}
private void run() throws Exception {
    Security.addProvider(new com.sun.crypto.provider.SunJCE());
    //Alice 生成 DH 对
    System.out.println("Alice create DH pairs");
    KeyPairGenerator AliceKpairGen = KeyPairGenerator.getInstance("DH");
    AliceKpairGen.initialize(1024);
    KeyPair AliceKpair = AliceKpairGen.generateKeyPair();
    // Alice 生成公共密钥 AlicePubKeyEnc 并发送给 Bob,内含 DH 对的种子
    // Bob 接收到 Alice 的编码后的公钥,将其解码
    byte[] AlicePubKeyEnc = AliceKpair.getPublic().getEncoded();
    KeyFactory BobKeyFac = KeyFactory.getInstance("DH");
    X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec (AlicePubKeyEnc);
        PublicKey AlicePubKey = BobKeyFac.generatePublic(x509KeySpec);
        System.out.println("Bob get Alice\'s Pubkey");
        // Bob 还要读出种子参数,再用这个参数初始化他的 DH 对
```

```
        DHParameterSpec dhParamSpec = ((DHPublicKey)AlicePubKey).getParams();
        KeyPairGenerator BobKpairGen = KeyPairGenerator.getInstance("DH");
        BobKpairGen.initialize(dhParamSpec);
        KeyPair BobKpair = BobKpairGen.generateKeyPair();
        System.out.println("Bob create DH pairs");
        KeyAgreement BobKeyAgree = KeyAgreement.getInstance("DH");
        BobKeyAgree.init(BobKpair.getPrivate());
        System.out.println("Bob initial session key generator by his private key");
        //Bob 生成与 Alice 共享的会话密钥 BobSessionKey
        BobKeyAgree.doPhase(AlicePubKey, true);
        SecretKey BobSessionKey = BobKeyAgree.generateSecret("DES");
        System.out.println("Bob create session key successfully");
        // Bob 生成自己的公钥包并发送给 Alice
        byte[] BobPubKeyEnc = BobKpair.getPublic().getEncoded();
        System.out.println("Bob send his public key to Alice");
        // Alice 接收到 BobPubKeyEnc 后解码出 Bob 的公钥
        KeyFactory AliceKeyFac = KeyFactory.getInstance("DH");
        x509KeySpec = new X509EncodedKeySpec(BobPubKeyEnc);
        PublicKey BobPubKey = AliceKeyFac.generatePublic(x509KeySpec);
        System.out.println("Alice get Bob's public key");
        //Alice 用自己的私钥初始化会话密钥生成器,同时验证 Bob 的身份
        KeyAgreement AliceKeyAgree = KeyAgreement.getInstance("DH");
        AliceKeyAgree.init(AliceKpair.getPrivate());
        System.out.println("Alice initialize Alice's session key generator");
        AliceKeyAgree.doPhase(BobPubKey, true);
        // Alice 生成会话密钥
        SecretKey AliceSessionKey = AliceKeyAgree.generateSecret("DES");
        System.out.println("Alice create session key successfully");
        // 现在 Alice 与 Bob 有了相同的会话密钥
        if (AliceSessionKey.equals(BobSessionKey)) System.out.println("Now, Alice & Bob have
the same key");
        // Bob 用 BobSessionKey 密钥加密信息
        Cipher BobCipher = Cipher.getInstance("DES");
        BobCipher.init(Cipher.ENCRYPT_MODE, BobSessionKey);
        String BobMsg = "TestOfDHKey";
        System.out.println("Bob's plaintext: " + BobMsg);
        byte[] cleartext = BobMsg.getBytes();
        byte[] ciphertext = BobCipher.doFinal(cleartext);
        // Alice 用 AliceSessionKey 密钥解密
        Cipher AliceCipher = Cipher.getInstance("DES");
        AliceCipher.init(Cipher.DECRYPT_MODE, AliceSessionKey);
        byte[] recovered = AliceCipher.doFinal(ciphertext);
        System.out.println("Alice decrypt Bob's cipher: " + (new String(recovered)));
        if (!java.util.Arrays.equals(cleartext, recovered)) throw new Exception("解密后与原
文信息不同");
        System.out.println("解密成功");
        }
    }
```

程序运行结果如下:

Alice create DH pairs

```
Bob get Alice's Pubkey
Bob create DH pairs
Bob initial session key generator by his private key
Bob create session key successfully
Bob send his public key to Alice
Alice get Bob's public key
Alice initialize Alice's session key generator
Alice create session key successfully
Now,Alice & Bob have the same key
Bob's plaintext:TestOfDHKey
Alice decrypt Bob's cipher:TestOfDHKey
解密成功。
```

10.3 JSSE

在 JDK 1.4 版本之前,JSSE 是一个可选的包。从 JDK 1.4 版本开始,JSSE 成为了标准的 JavaAPI。

JSSE API 通过提供扩充的网络 Socket 类、证书管理器、密钥管理器、SSLContext 和用来封装 Socket 创建的 Factory 框架,完善了 java.security 包和 java.net 包中的核心加密服务。

JSSE 的 API 支持 SSL 版本 2.0、3.0 和 TLS 版本 1.0。

10.3.1 SSL

SSL 编程需使用以下几个 java 包。

(1) javax.net.ssl:包括进行安全通信的类,如 SSLServerSocket 和 SSLSocket 类。

(2) javax.net:包括 SSL 的 Factory 类,如 SSLServerSocketFactory 和 SSLSocketFactory 类。

(3) java.security.cert:包括处理安全证书的类,如 X509Certificate 类。

(4) com.sun.net.ssl:包括 SUN 公司提供 JSSE 的实现类。

JSSE 中的核心类是 SSLServerSocket 类和 SSLSocket 类,它们分别是 ServerSocket 和 Socket 类的子类。

SSLServerSocket 对象由 SSLServerSocketFactory 创建,SSLSocket 对象由 SSLSocket Factory 创建。而 SSLSocketFactory、SSLServerSocketFactory 以及 SSLEngine 对象又都由 SSLContext 对象创建。

下面通过一个 Client/Server 编程实例来介绍如何利用 JSSE 进行 SSL 编程。

建立 SSL 的服务器和客户端时需要用到密钥库,因此在编程前应将密钥库准备好,这里可以直接使用 10.1.3 节中的密钥库:keystore。

建立 SSL 服务器的具体步骤如下:

(1) 设置密钥库及口令属性。javax.net.ssl.keyStore 指定密钥库的名称,javax.net.

ssl. keyStorePassword 指定密钥库的密码。

```
System.setProperty("javax.net.ssl.keyStore",".keystore");
System.setProperty("javax.net.ssl.keyStorePassword","abc123");
```

（2）创建 SSLServerSocketFactory 对象。

```
SSLServerSocketFactory sslssf = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault( );
```

（3）创建 ServerSocket 对象，此时需要指定服务的端口号。

```
ServerSocket servsock = sslssf.createServerSocket(9632);
```

（4）等待客户的连接请求。

```
Socket socket1 = servsock.accept( );
```

（5）连接建立以后，可以向客户端发送信息。

```
PrintStream out  =  new PrintStream(socket1.getOutputStream( ));
out.println("Hi client, It's server.");
```

以下是建立 SSL Server 的完整程序：

```
import java.net.*;
import java.io.*;
import javax.net.ssl.*;
public class TestSimpleSSLServer{
/*
 *     @author Ray
 */
public static void main(String args[ ]) throws Exception{
    System.setProperty("javax.net.ssl.keyStore",".keystore");
    System.setProperty("javax.net.ssl.keyStorePassword","abc123");
    SSLServerSocketFactory sslssf = (SSLServerSocketFactory)
    SSLServerSocketFactory.getDefault( );
    ServerSocket servsock = sslssf.createServerSocket(9632);
    System.out.println("Server is ready…");
    while(true){
        Socket socket1 = servsock.accept( );
        PrintStream out  =  new PrintStream(socket1.getOutputStream( ));
        out.println("Hi client, It's server.");
        out.close( );
        socket1.close( );
    }
}
}
```

运行结果如图 10.5 所示。

建立客户端之前，需要将服务器的公钥证书先放在客户端一个密钥库中，并在程序中指定。

这里将 10.1.3 小节中的证书 ray20120220.cer 导入到密钥库 clienttrust 中，相应的语句是 keytool -import-alias ray20120220-file ray20120220.cer-keystore clienttrust。

```
C:\WINDOWS\system32\cmd.exe - java TestSimpleSS...
D:\java\JavaSecurity>javac TestSimpleSSLServer.java

D:\java\JavaSecurity>java TestSimpleSSLServer
Server is ready……
```

图 10.5 SSL Server 运行结果

具体步骤如下：

(1) 设置客户程序信任的密钥库。

```
System.setProperty("javax.net.ssl.trustStore", "clienttrust");
```

因为 clienttrust 中存放的是公钥证书，所以不需要密码。

(2) 创建 SSLSocketFactory 对象。

```
SSLSocketFactory sslssf = (SSLSocketFactory) SSLSocketFactory.getDefault();
```

(3) 创建 Socket 对象，建立连接。

```
Socket socket1 = sslssf.createSocket("192.168.1.98", 9632);
```

这里的端口号要与服务器的 SSL 服务端口号一致。

(4) 连接建立以后，可以接收来自服务器端的信息。

```
BufferedReader in = new BufferedReader(new InputStreamReader(socket1.getInputStream()));
String line = in.readLine();
```

当然客户端也可以用同样的方法向服务器发送信息。

实现 SSL Client 的完整程序如下：

```
import java.net.*;
import java.io.*;
import javax.net.ssl.*;
public class TestSimpleSSLClient{
/*
 *     @author Ray
 */
public static void main(String args[]) throws Exception {
      System.setProperty("javax.net.ssl.trustStore", "clienttrust");
      SSLSocketFactory sslssf = (SSLSocketFactory) SSLSocketFactory.getDefault();
      Socket socket1 = sslssf.createSocket("192.168.1.98", 9632);
      BufferedReader in = new BufferedReader(new InputStreamReader
      (socket1.getInputStream()));
      String line = in.readLine();
      System.out.println(line);
      in.close();
}
}
```

运行结果如图 10.6 所示。

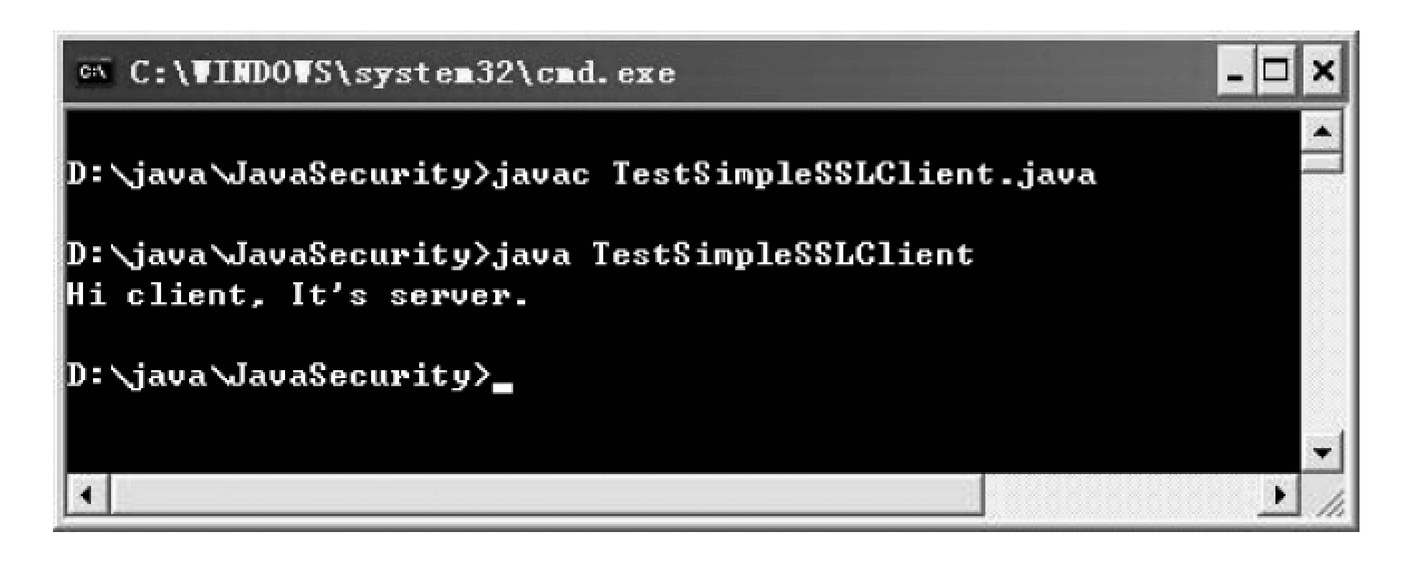


图 10.6　SSL Client 运行结果

10.3.2　HTTPS

HTTPS 是最常见的 SSL 应用，下面将用 JSSE 来建立一个简单的 HTTPS 服务器。

建立 HTTPS 服务器的过程与建立一般的 SSL 服务器过程类似，特别之处包括以下几个方面：

(1) 由于服务器既要接收客户端的请求又要给出响应，所以通常既要获取输入流又要获取输出流。

通过 Socket 的 getOutputStream()方法可以得到 OutputStream 对象，通过 Socket 对象的 getInputStream()方法可以创建 BufferedReader 类型的对象。

相应的语句是：

```
PrintStream out = new PrintStream(socket1.getOutputStream());
BufferedReader in = new BufferedReader(new InputStreamReader(socket1.getInputStream()));
```

(2) 向客户端发送的响应信息需要符合 HTTP 协议的要求，如根据 HTTP 协议的规定，Web 服务器收到浏览器发来的请求后，会将浏览器 HTTP 版本、数据类型等信息发送给客户端。

相应的语句是：

```
out.println("HTTP/1.0 200 OK");
out.println("MIME_version:1.0");
out.println("Content_Type:text/html");
```

在接下来的实例中，将实现一个简单的网站访问计数功能。

以下是 HTTPS Server 的完整程序：

```
import java.net.*;
import java.io.*;
import javax.net.ssl.*;
public class TestSimpleHTTPSServer {
/*
 *     @author Ray
 */
        public static void main(String args[]) {
                int count = 0;
                try {
```

```
                System.setProperty("javax.net.ssl.keyStore",".keystore");
                System.setProperty("javax.net.ssl.keyStorePassword","abc123");
                SSLServerSocketFactory sslssf = (SSLServerSocketFactory)
                SSLServerSocketFactory.getDefault();
                ServerSocket servsock = sslssf.createServerSocket(443);
                System.out.println("Web server is ready …");
                while(true){
                    Socket socket1 = servsock.accept(); //等待请求
                    PrintStream out = new PrintStream(socket1.getOutputStream());
                    BufferedReader in = new BufferedReader(new
                    InputStreamReader(socket1.getInputStream()));
                    String msg = null;
                    while(( msg = in.readLine())! = null){
                    //读取客户端信息
                    System.out.println("Get message \"" + msg + "\" from client.");
                    if(msg.equals("")) break;
                    }
                    out.println("HTTP/1.0 200 OK\r\nContent - Type: text/html\r\n");
                    //显示计数
                    out.println("<html><head></head><body> You are welcome, Dear
                    No. " + (count++) + " visitor.</Body></html>");
                    out.close();
                    socket1.close();
                    in.close();
                }
            } catch (IOException e) {
            System.out.println(e);
            }
        }
}
```

运行结果如图 10.7 所示。

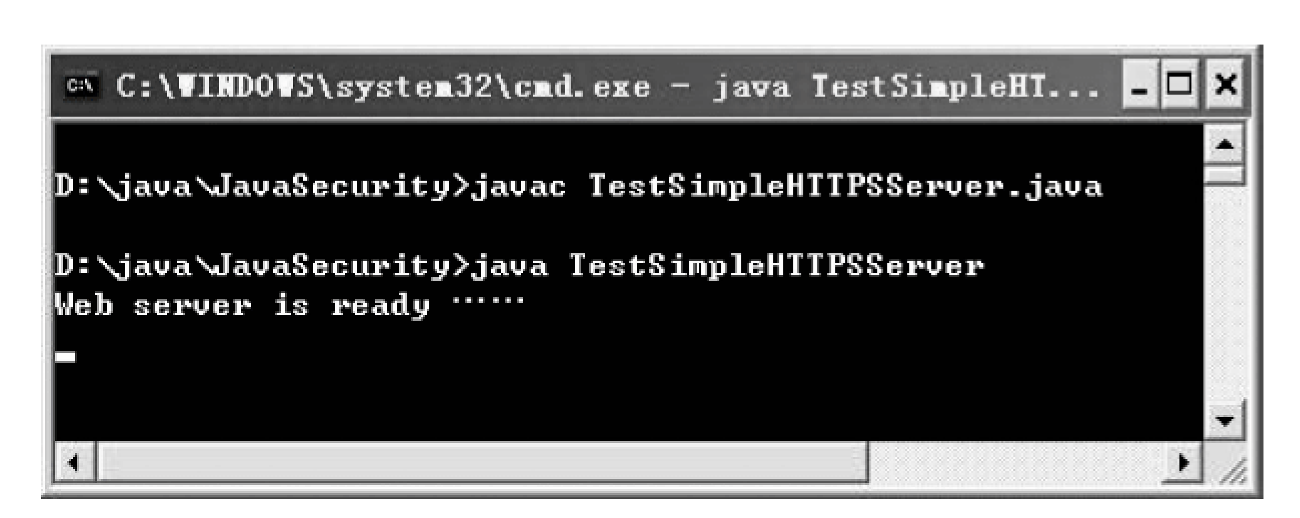


图 10.7 HTTPS 服务器运行结果

通过浏览器访问该服务器,结果如图 10.8 所示。

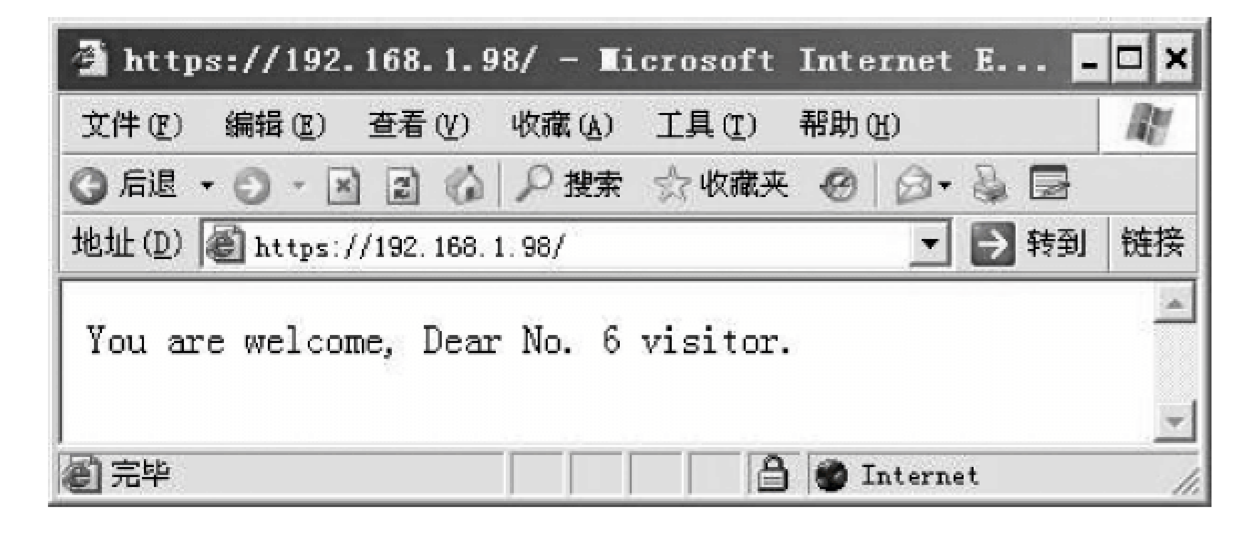


图 10.8 浏览器访问 HTTPS 服务器运行结果

此时服务器端可以看到客户端的相关信息，如图 10.9 所示。

```
C:\WINDOWS\system32\cmd.exe - java TestSimpleHTTPSServer
D:\java\JavaSecurity>java TestSimpleHTTPSServer
Web server is ready ……
Get message "GET / HTTP/1.1" from client.
Get message "Accept: */*" from client.
Get message "Accept-Language: zh-cn" from client.
Get message "Accept-Encoding: gzip, deflate" from client.
Get message "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
" from client.
Get message "Host: 192.168.1.98" from client.
Get message "Connection: Keep-Alive" from client.
Get message "" from client.
Get message "GET / HTTP/1.1" from client.
```

图 10.9　HTTPS 服务器运行结果

如果不用浏览器访问，还可以自己编写客户端程序。

其程序如下：

```
import java.net.*;
import java.io.*;
import javax.net.ssl.*;
public class TestSimpleHTTPSClient {
      public static void main(String args[])throws Exception {
            try {
            /*
             *     @author Ray
             */
                  System.setProperty("javax.net.ssl.trustStore", "clienttrust");
                  String hostname = "192.168.1.98";
                  int port = 443;
                  SSLSocketFactory sslssf = (SSLSocketFactory)
                  SSLSocketFactory.getDefault();
                  Socket socket1 = sslssf.createSocket(hostname, port);//建立连接
                  OutputStream outstrm = socket1.getOutputStream();
                  PrintStream out = new PrintStream(outstrm);
                  InputStream instrm = socket1.getInputStream();
                  BufferedReader in = new BufferedReader(new InputStreamReader(instrm));
                  //向服务器发送信息
                  out.println("Hi HTTPS Server, It's client.");
                  out.println("");
                  String line = null;
                  while((line = in.readLine())! = null){
                      System.out.println(line);
                  }
                  in.close();
                  out.close();
            } catch(IOException e) {
            }
      }
}
```

程序运行结果如图 10.10 所示。



图 10.10　Java 程序访问 HTTP 服务器运行结果

10.4　JAAS

JAAS 提供了根据用户标识来限制资源访问的认证标准方法。

JAAS 认证采用可插入式验证模块(PAM),允许应用程序同底层认证技术的具体实现保持独立,新增或者更新认证方法并不需要更改应用程序。

JAAS 认证的核心类和接口包括以下内容。

(1) LoginContext:应用程序通过实例化 LoginContext 对象开始认证过程。

(2) LoginModule:引用配置文件中的具体认证方法,即 LoginModule 对象,来执行认证。

(3) CallbackHandler,Callback:callback 包中的 CallbackHandler 类要求用户输入用户信息,如别名、密钥库密码和私钥密码等。

使用 JAAS 进行认证的具体步骤如下:

(1) 创建和用户交互的回调处理器对象。

```
DialogCallbackHandler handler = new DialogCallbackHandler();
```

除了 DialogCallbackHandler 类外,callback 包还提供了 TextCallbackHandler,以文本方式和用户交互。

(2) 创建 LoginContext 对象。

```
LoginContext lc = new LoginContext("ray20120224",handler);
```

LoginContext 类的构造器有两个参数,一个是配置文件中的条目名称;另一个参数是前面创建的回调处理器。

(3) 执行登录操作,相应的语句是:

```
lc.login();
```

(4) 登录成功后,可进行进一步的处理。如:

```
Subject subj = lc.getSubject();
System.out.println(subj.getPrincipals());
```

通过 getSubject()方法可得到代表登录主体名称,通过 getPrincipals()方法可得到身份信息。

(5) 如果登录不成功,则可以处理 LoginException 对象,返回出错信息或重新登录。

以下是一个完整的 JAAS 认证程序:

```
import com.sun.security.auth.callback.DialogCallbackHandler;
import javax.security.auth.*;
import javax.security.auth.login.*;
public class TestSimpleLogin {
        public static void main(String[] args) throws Exception {
        /*
         * @author Ray
         */
            //指定登录配置文件
            DialogCallbackHandler handler = new DialogCallbackHandler();
            LoginContext lc = new LoginContext("ray20120224",handler);
            boolean passflag;
            try {
                    lc.login();
                    //登录成功
                    passflag = true;
                    System.out.println("Login succeeded!");
                    Subject subj = lc.getSubject();
                    System.out.println(subj.getPrincipals());
            }
            catch (LoginException le) {
                    //登录失败
                    passflag = false;
                    System.err.println("Login failed!");
            }
        }
}
```

相应的配置文件 ray20120224.config,内容如下:

```
ray20120224 {
            com.sun.security.auth.module.KeyStoreLoginModule required
            keyStoreURL = "file:D:/java/javaSecurity/.keystore";
}
```

运行时,输入命令:

```
java -Djava.security.auth.login.config = ray20120224.config TestSimpleLogin
```

将弹出一个登录界面,如图 10.11 所示。

如果验证成功,则输出结果如图 10.12 所示。

Confirmation
请输入密钥库信息
密钥库别名:
密钥库口令:
私有密钥口令 (可选):
确定 取消

图 10.11 登录窗口



图 10.12 登录成功后的输出结果

以上的实例要求用户输入密钥库信息进行验证，这是一种默认的方式。如果需要编写其他风格的登录模块，只要编写类重新实现 LoginModule 接口中的 initialize()、login()、commit()、abort()和 logout()等方法即可。

10.5 习 题

1. Java 加密体系结构包括哪几个部分?
2. 在 Java 加密体系结构中，________提供基于 SSL 的安全通信功能。
3. 在 Java 加密体系结构中，________提供用户身份认证的功能。
4. 在 java.security 包中的________类提供了计算消息摘要的服务。
5. RSA 密码算法的参数往往非常大，所以在运算中一般需要使用________类。
6. 编程实现一个 HTTPS 服务器。
7. 编程实现一个 SSL 客户端。
8. 编程实现一个基于 JAAS 的登录模块。
9. 分析以下程序段的目的，试求相应的运行结果。

```
String key = "ipadipad";
SecretKeySpec sks = new SecretKeySpec(key.getBytes(),"HmacMD5");
Mac mac1 = Mac.getInstance("HmacMD5");
mac1.init(sks);
String msg = "Helloworld";
mac1.update(msg.getBytes());
System.out.println(byte2hex(mac1.doFinal()));
```

附录 A　Shannon 理论

1949 年，Claude Shannon 在 *Bell Systems Technical Journal* 上发表了题为 *Communication Theory of Secrecy Systems* 的论文。这篇论文对密码学的发展产生了巨大的影响，使密码学从根据经验来研究转变到根据科学来研究，其中最重要的理论就是本附录要讨论的 Shannon 信息论。

A.1　概率论基础

密码体制的无条件安全性不能用计算复杂性的观点来研究，谈及无条件安全，允许的计算时间是无限的。

研究无条件安全的合适框架是概率论。

定义 A.1（随机变量）　一个离散的随机变量，由有限集合 X 和定义在 X 上的概率分布组成。

用 $\Pr[X=x]$ 表示随机变量 X 取 x 时的概率，如果随机变量是固定的，也可缩写成 $\Pr[X]$。对任意的 $x \in X$，则有：

$$0 \leqslant \Pr[x] \leqslant 1$$

并且：

$$\sum_{x \in X} \Pr[x] = 1$$

例如，可以把抛硬币看成定义在集合{heads，tails}上的随机变量。相关的概率分布可以是：

$$\Pr[\text{heads}]=\Pr[\text{tails}]=1/2$$

假设 X 是定义在集合 X 上的随机变量，$E \subseteq X$。x 在 X 子集 E 上取值的概率计算如下：

$$\Pr[x \in E] = \sum_{x \in E} \Pr[x]$$

子集 E 通常称为事件。

定义 A.2（独立性）　假设 X 和 Y 是分别定义在有限集合 X 和 Y 上的随机变量。

联合概率 $\Pr[x,y]$是 X 取 x 并且 Y 取 y 的概率；条件概率 $\Pr[x|y]$表示 Y 取 y 时 X 取 x 的概率。

如果对任意的 $x \in X$ 和 $y \in Y$，成立：

$$\Pr[x,y]=\Pr[x]\Pr[y]$$

称随机变量 X 和 Y 是统计独立的。

联合概率和条件概率可以通过下面的等式和贝叶斯定理联系起来。

$$\Pr[x,y]=\Pr[x|y]\Pr[y]$$

$$\Pr[x,y]=\Pr[y|x]\Pr[x]$$

定理 A.1(贝叶斯定理) 如果 $\Pr[y]>0$,那么:

$$\Pr[x \mid y] = \frac{\Pr[x]\Pr[y \mid x]}{\Pr[y]}$$

推论 A.1 X 和 Y 是统计独立的随机变量,当且仅当对所有的 $x\in X$ 和 $y\in Y$,成立:

$$\Pr[x|y]=\Pr[x]$$

A.2 完善保密性

假设(P,C,K,E,D)是一个特定的密码体制,密钥只使用一次。明文空间 P 存在一个概率分布,可定义一个随机变量用 X 表示。$\Pr[X=x]$表示明文 x 发生的先验概率。

还假设 Alice(发送者)和 Bob(接收者)以固定的概率分布选取密钥(通常密钥是随机选取的,因此所有的密钥都是等概率的)。

密钥也定义一个随机变量,用 K 表示。$\Pr[K=k]$表示密钥 k 发生的概率。

密钥是在 Alice 知道明文之前选取的,因此也可以合理地假设密钥与明文是独立的随机变量。

P 和 K 的概率分布可导出 C 的概率分布。同样可以把密文看成随机变量,用 y 表示。密文 y 的概率表示为 $\Pr[Y=y]$,定义:

$$C(k) = \{e_k(x): x \in P\}$$

$C(k)$代表密钥是 k 时所有可能的密文。对于任意的 $y\in C$,则有:

$$\Pr[Y = y] = \sum_{\{K: y\in C(k)\}} \Pr[K = k]\Pr[X = d_k(y)]$$

对于任意的 $y\in C$ 和 $x\in P$,可计算条件概率 $\Pr[Y=y|X=x]$(也就是给定明文 x,密文 y 的概率):

$$\Pr[Y = y \mid X = x] = \sum_{\{K: x=d_k(y)\}} \Pr[K = k]$$

根据贝叶斯定理计算条件概率 $\Pr[X=x|Y=y]$:

$$\Pr[X = x \mid Y = y] = \frac{\Pr[X = x]\times \sum_{\{k: x=d_k(y)\}} \Pr[K = k]}{\sum_{\{k: y\in C(k)\}} \Pr[K = k]\Pr[X = d_k(y)]}$$

只要知道概率分布便可以完成以上计算。

例 A.1 假设 $P=\{a,b\}$满足:

$$\Pr[a] = 1/4,\quad \Pr[b] = 3/4$$

设 $K=\{k_1,k_2,k_3\}$

$$\Pr\{k_1\} = 1/2,\quad \Pr\{k_2\} = \Pr\{k_3\} = 1/4$$
$$C = \{1,2,3,4\}$$

加密函数定义为:

$$e_{k_1}(a) = 1,\quad e_{k_1}(b) = 2,\quad e_{k_2}(a) = 2,\quad e_{k_2}(b) = 3,\quad e_{k_3}(a) = 3,\quad e_{k_3}(b) = 4$$

以上加密函数也可以用表 A.1 表示。

表 A.1 加密表

	a	b
k_1	1	2
k_2	2	3
k_3	3	4

计算 e 上的概率分布如下：

$$\Pr[1]=\frac{1}{8}$$

$$\Pr[2]=\frac{3}{8}+\frac{1}{16}=\frac{7}{16}$$

$$\Pr[3]=\frac{3}{16}+\frac{1}{16}=\frac{1}{4}$$

$$\Pr[4]=\frac{3}{16}$$

接下来计算给定密文条件下，明文空间上的条件概率分布为：

$$\Pr[a \mid 1]=1 \quad \Pr[b \mid 1]=0$$

$$\Pr[a \mid 2]=\frac{1}{7} \quad \Pr[b \mid 2]=\frac{6}{7}$$

$$\Pr[a \mid 3]=\frac{1}{4} \quad \Pr[b \mid 3]=\frac{3}{4}$$

$$\Pr[a \mid 4]=0 \quad \Pr[b \mid 4]=1$$

然后定义完善保密性的概念。简单地讲，完善保密性就是说攻击者(Eve)不能通过观察密文得到明文的任何信息。

定义 A.3(**完善保密性**) 一个密码体制具有完善保密性，即如果对于任意的 $x\in P$ 和 $y\in C$，则有：

$$\Pr[x \mid y]=\Pr[x]$$

也就是说，给定密文 y，明文为 x 的后验概率等于明文 x 的先验概率。

在例 A.1 中，对于密文 $y=3$ 满足完善保密性的定义，但是对其他的密文不满足。

现在证明移位密码的完善保密性。从直觉上看这是很显然的，因为对于任意给定的密文 $y\in Z_{26}$，任何 $x\in Z_{26}$ 都可能是对应的明文。

定理 A.2 假设移位密码的 26 个密钥都是以相同的概率 1/26 使用的，则对任意的明文概率分布，移位密码具有完善保密性。

证明：这里 $P=C=K=Z_{26}$，对于 $0\leqslant k\leqslant 25$，加密函数 e_k 定义为：

$$e_k(x)=(x+k)\bmod 26$$

首先计算 C 上的概率分布。

假设 $y\in Z_{26}$，则：

$$\begin{aligned}\Pr[Y=y]&=\sum_{k\in Z_{26}}\Pr[K=k]\Pr[X=d_k(y)]\\&=\sum_{k\in Z_{26}}\frac{1}{26}\Pr[X=y-k]=\frac{1}{26}\sum_{k\in Z_{26}}\Pr[X=y-k]\end{aligned}$$

如果固定 y，值$(y-k) \bmod 26$ 构成一个置换。因此：

$$\sum_{k\in Z_{26}} \Pr[X=y-k]=\sum_{k\in Z_{26}} \Pr[X=x]=1$$

得到对于任意的 $y\in Z_{26}$，则：

$$\Pr[y]=\frac{1}{26}$$

接下来对于任意的 x，y，有：

$$\Pr[y\mid x]=\Pr[k=(y-x) \bmod 26]=\frac{1}{26}$$

应用贝叶斯定理，很容易计算出：

$$\Pr[x\mid y]=\frac{\Pr[x]\Pr[y\mid x]}{\Pr[y]}$$

$$=\frac{\Pr[x]\frac{1}{26}}{\frac{1}{26}}=\Pr[x]$$

所以这个密码体制具有完善保密性。

下面考察一般情况下的完善保密性。

使用贝叶斯定理，对于所有的 $x\in P$ 和 $y\in C$，则有：

$$\Pr[x\mid y]=\Pr[x]$$

等价于对于所有的 $x\in P$ 和 $y\in C$，则有：

$$\Pr[y\mid x]=\Pr[y]$$

这里可以合理地假设对于所有的 $y\in C$，$\Pr[y]>0$(如果 $\Pr[y]=0$，则密文 y 不会出现，可以从 E 中去掉)。

任意固定的 $x\in P$，对于任意 $y\in C$，则有：

$$\Pr[y\mid x]=\Pr[y]>0$$

成立。

因此，对于任意 $y\in C$，一定至少存在一个密钥 K 满足 $e_k(x)=y$。这样有 $|K|\geqslant|C|$。

在一个密码体制中，因为加密函数是单射，一定有 $|C|\geqslant|P|$。

定理 A.3 假设密码体制(P,C,K,E,D)满足 $|K|=|C|=|P|$，这个密码体制是完善保密的。当且仅当每个密钥被使用的概率都是 $1/|K|$，并且对于任意的 $x\in P$ 和 $y\in C$，存在唯一的密钥 K 使得 $e_k(x)=y$。

证明：假设这个密码体制是完善保密的。由以上可知，对于任意的 $x\in P$ 和 $y\in C$，至少存在一个密钥 K 使得 $e_k(x)=y$。因此不等式：

$$|C|=|\{e_k(x):k\in K\}|\leqslant|K|$$

成立。

但是这里假设 $|C|=|K|$，因此一定有：

$$|\{e_k(x):k\in K\}|\leqslant|K|$$

也就是说，不存在两个不同的密钥 k_1 和 k_2 使得 $e_{k_1}(x)=e_{k_2}(x)=y$；所以对于 $x\in P$ 和 $y\in C$，刚好存在一个密钥 k 使得 $e_k(x)=y$。

记 $n=|K|$，设 $P=\{x_i:1\leqslant i\leqslant n\}$ 并且固定一个密文 $y\in C$。设密钥为 k_1、k_2、…、k_n，并且

$e_{k_i}(x_i)=y, 1\leqslant i\leqslant n$。运用贝叶斯定理可得：

$$\Pr[x_i \mid y]=\frac{\Pr[y \mid x_i]\Pr[x_i]}{\Pr[y]}=\frac{\Pr[k=k_i]\Pr[x_i]}{\Pr[y]}$$

考虑到完善保密的条件 $\Pr[x_i|y]=\Pr[x_i]$，这里 $\Pr[k_i]=\Pr[y], 1\leqslant i\leqslant n$，也可以说所有的密钥都是等概率使用的（概率为 $\Pr[y]$）。

密钥的数目为 K，于是可得对任意的 $k\in K, \Pr[k]=1/|K|$。

相反地，如果两个假设的条件都成立，类似于定理 A.2 的证明，可以很容易得到密码体制是完善保密的。

A.3　熵的概念

在密码学的发展历史中，人们试图设计出密钥可以加密相对长的明文（即用一个密钥加密多个消息），并且仍然可以保持一定的计算安全性。下面将讨论用一个密钥加密多个消息会有什么后果，并且还要讨论如果给密码分析员足够的时间，进行一次成功的唯密文攻击有多大的可能性。

研究这个问题的基本工具是熵（Entropy）。熵的概念来自于 Shannon 于 1948 年创建的信息论。熵可以看作是对信息或不确定性的数学度量，通过概率分布的函数来计算。

假设离散的随机变量 X 根据特定的概率分布从有限集合 X 中取值。可以从概率分布的实验结果中得到什么信息呢？或者说，在实验发生之前，结果的不确定性如何呢？这个性质称为 X 的熵，用 $H(X)$ 表示。

下面给出个具体的例子描述熵的概念。

假设随机变量 X 代表掷硬币，相关的概率分布是 $\Pr[\text{heads}]=\Pr[\text{tails}]=1/2$。

既然可以将“heads”编码为 1，将“tails”编码为 0，那么一次掷硬币的信息或者熵是 1 比特。用同样的方式，n 次独立的掷硬币的熵是 n 比特，即 n 次投掷可以用长为 n 比特的比特串进行编码。

下面给出稍微复杂一点的例子。

假设一个随机变量 X，取 3 种可能的值 x_1、x_2、x_3，概率分别为 1/2、1/4、1/4。对这 3 种结果最有效的编码是将 x_1 表示成 0，x_2 表示成 10，x_3 表示成 11。那么编码的平均比特长度是：

$$\frac{1}{2}\times 1+\frac{1}{4}\times 2+\frac{1}{4}\times 2=\frac{3}{2}$$

上面的例子说明以概率 2^{-n} 发生的事件可以编码成长度为 n 的比特串。一般地，可以想象以概率 p 发生的结果可以编码成长大约为 $-\log_2 p$ 的比特串。对于任意的概率分布为 p_1、p_2、…、p_n 的随机变量 X，将 $-\log_2 p_i$ 的加权平均值作为对信息的度量。

定义 A.4　（**熵**）假定随机变量 X 在有限集合 X 上取值，则随机变量 X 的熵定义为：

$$H(X)=-\sum_{x\in X}\Pr[x]\log_2\Pr[x]$$

对于熵的定义，有以下几点需注意：

考虑到 $y=0$ 时 $\log_2 y$ 没有意义，熵有时定义为在所有非零的概率上的和。但是，由于

$$\lim_{y \to 0} \log_2 y = 0$$

对于某些 x 可以令 $\Pr[x]=0$。

同时考虑到选择 2 作为指数的底也是任意的,选取其他的底仅仅改变了熵的常数因子。

如果 $|X|=n$ 并且对于所有的 $x \in X, \Pr[x]=1/n$,那么 $H(X)=\log_2 n$。

同样,容易知道对于任意的随机变量 $X, H(X) \geqslant 0$。

$H(X)=0$ 当且仅当对于某一个 $x_0 \in X, \Pr[x_0]=1$,对于所有的 $x \neq x_0, \Pr[x]=0$。

下面考虑密码体制中不同部分的熵,把密钥看成在 K 上取值的随机变量 k,可以计算熵 $H(k)$。同样地,可以分别计算同明文和密文相关的熵 $H(P)$ 和 $H(C)$。

为了更好地表示密码体制中的熵的计算方法,下面给出 A.2 节中例 A.1 的熵。

例 A.2 熵的计算。

$$\begin{aligned} H(P) &= -\frac{1}{4} \times \log_2 \frac{1}{4} - \frac{3}{4} \times \log_2 \frac{3}{4} \\ &= -\frac{1}{4} \times (-2) - \frac{3}{4} \times (\log_2 3 - 2) \\ &= 2 - \frac{3}{4} \times \log_2 3 \approx 0.81 \end{aligned}$$

同理,可以计算得出 $H(K)=1.5, H(C) \approx 1.85$。

A.4 熵的性质

本节主要介绍一些与熵有关的基本结论的证明。

首先描述一个基本结论,称为 Jensen 不等式。这个不等式对熵的讨论比较重要,Jensen 不等式涉及凸函数,下面给出其定义。

定义 A.5 (**凸函数**)区间 I 上的实值函数 f 是凸函数,如果对任意的 x、$y \in I$ 满足:

$$f\left(\frac{x+y}{2}\right) \geqslant \frac{f(x)+f(y)}{2}$$

f 是区间 I 上严格的凸函数,如果对任意的 x、$y \in I$, $x \neq y$ 满足:

$$f\left(\frac{x+y}{2}\right) > \frac{f(x)+f(y)}{2}$$

定理 A.4 (**Jensen 不等式**)假设 f 是区间 I 上连续的严格的凸函数

$$\sum_{i=1}^{n} a_i = 1$$

其中,$a_i > 0, 1 \leqslant i \leqslant n$,那么:

$$\sum_{i=1}^{n} a_i f(x_i) \leqslant f\left(\sum_{i=1}^{n} a_i x_i\right)$$

其中,$x_i \in I, 1 \leqslant i \leqslant n$。当且仅当 $x_1 = x_2 = \cdots = x_n$ 成立时,等式成立。

定理 A.5 假设 X 是一个随机变量,概率分布为 p_1、p_2、$\cdots$、p_n,其中,$p_i > 0$, $1 \leqslant i \leqslant n$;那么:

$$H(X) \leqslant \log_2 n$$

当且仅当 $p_i=1/n$，$1\leqslant i\leqslant n$ 时，等式成立。

证明：根据 Jensen 不等式：

$$\begin{aligned}H(X)&=-\sum_{i=1}^{n}p_i\log_2 p_i\\&=\sum_{i=1}^{n}p_i\log_2\frac{1}{p_i}\\&\leqslant\log_2\sum_{i=1}^{n}\left(p_i\times\frac{1}{p_i}\right)\\&=\log_2 n\end{aligned}$$

当且仅当 $p_i=1/n$，$1\leqslant i\leqslant n$ 时，等式成立。命题得证。

定理 A.6　$H(X,Y)\leqslant H(X)+H(Y)$，当且仅当 X 和 Y 统计独立时，等式成立。

证明：假设 X 取值 x_i，$1\leqslant i\leqslant m$，Y 取值 y_j，$1\leqslant j\leqslant n$。记 $p_i=\Pr[X=x_i]$，$1\leqslant i\leqslant m$；$q_j=\Pr[Y=y_j]$，$1\leqslant j\leqslant n$。记 $r_{ij}=\Pr[X=x_i,Y=y_j]$，$1\leqslant i\leqslant m$，$1\leqslant j\leqslant n$（这是联合概率分布）。

可得：

$$p_i=\sum_{j=1}^{n}r_{ij}(1\leqslant i\leqslant m),\quad q_j=\sum_{i=1}^{m}r_{ij}(1\leqslant j\leqslant n)$$

做如下计算：

$$\begin{aligned}H(X)+H(Y)&=-\left(\sum_{i=1}^{m}p_i\log_2 p_i+\sum_{j=1}^{n}q_j\log_2 q_j\right)\\&=-\left(\sum_{i=1}^{m}\sum_{j=1}^{n}r_{ij}\log_2 p_i+\sum_{j=1}^{n}\sum_{i=1}^{m}r_{ij}\log_2 q_j\right)\\&=-\sum_{i=1}^{m}\sum_{j=1}^{n}r_{ij}\log_2 p_iq_j\end{aligned}$$

另有：

$$H(X,Y)=-\sum_{i=1}^{m}\sum_{j=1}^{n}r_{ij}\log_2 r_{ij}$$

于是有：

$$\begin{aligned}H(X,Y)-H(X)-H(Y)&=\sum_{i=1}^{m}\sum_{j=1}^{n}r_{ij}\log_2\frac{1}{r_{ij}}+\sum_{i=1}^{m}\sum_{j=1}^{n}r_{ij}\log_2 p_iq_j\\&=\sum_{i=1}^{m}\sum_{j=1}^{n}r_{ij}\log_2\frac{p_iq_j}{r_{ij}}\\&\leqslant\log_2\sum_{i=1}^{m}\sum_{j=1}^{n}p_iq_j\\&=\log_2 1\\&=0\end{aligned}$$

当取等号时，存在常数 c 对于所有的 i、j 使得 $p_iq_j/r_{ij}=c$。利用：

$$\sum_{j=1}^{n}\sum_{i=1}^{m}r_{ij}=\sum_{j=1}^{n}\sum_{i=1}^{m}p_iq_j=1$$

于是有 $c=1$。因此，当且仅当 $r_{ij}=p_iq_j$ 时等式成立，也就是当且仅当：

$$\Pr[X=x_i,Y=y_i]=\Pr[X=x_i]\Pr[Y=y_i]$$

$1\leqslant i\leqslant m$，$1\leqslant j\leqslant n$。这说明 X 和 Y 是统计独立的。命题得证。

定义 A.6 （**条件熵**）假设 X 和 Y 是两个随机变量，对于 Y 的任何固定值 y，得到一个 X 上的（条件）概率分布，记相应的随机变量为 $X|y$。显然：

$$H(X\mid y)=-\sum_x \Pr[x\mid y]\log_2\Pr[x\mid y]$$

定义条件熵 $H(X|Y)$ 为熵 $H(X|y)$ 取所有 y 的加权平均值：

$$H(X\mid Y)=-\sum_y\sum_x \Pr[y]\Pr[x\mid y]\log_2\Pr[x\mid y]$$

条件熵度量了 Y 揭示的 X 的平均信息量。

根据以上定义和定理，显然还可以得到下面的定理和推论。

定理 A.7

$$H(X,Y)=H(Y)+H(X\mid Y)$$

推论 A.2 $H(X,Y)\leqslant H(X)$，当且仅当 X 和 Y 统计独立时，等式成立。

A.5 伪密钥和唯一解距离

条件熵 $H(K|C)$ 又称为密钥含糊度，可度量给定密文下密钥的不确定性。

定理 A.8 设 (P,C,K,E,D) 是一个密码体制，那么：

$$H(K\mid C)=H(K)+H(P)-H(C)$$

证明：根据密码系统的定义有：

$$H(K,P,C)=H(C\mid K,P)+H(K,P)$$

由于 $y=e_k(x)$，即密钥和明文唯一决定密文，可以得出：

$$H(C\mid K,P)=0$$

因此：

$$H(K,P,C)=H(K,P)$$

K 和 P 是统计独立的，因此有：

$$H(K,P)=H(K)+H(P)$$

于是：

$$H(K,P,C)=H(K,P)=H(K)+H(P)$$

同理，由于密钥和密文唯一决定明文，有 $H(P|K,C)=0$，可得：

$$H(K,P,C)=H(K,C)$$

进一步可以得出：

$$\begin{aligned}H(K\mid C)&=H(K,C)-H(C)\\&=H(K,P,C)-H(C)\\&=H(K)+H(P)-H(C)\end{aligned}$$

例 A.3 在 A.4 节中已经计算出：

$$H(P)\approx 0.81,\quad H(K)=1.5,\quad H(C)\approx 1.85$$

由定理 A.8 可计算出：

$$H(K \mid C) = H(K) + H(P) - H(C) \approx 1.5 + 0.81 - 1.85 \approx 0.46$$

通过条件熵的定义来计算出条件熵的值可以验证以上结果。

首先需要通过贝叶斯定理计算概率 $\Pr[K=k_i \mid y=j]$，$1 \leqslant i \leqslant 3$，$1 \leqslant j \leqslant 4$。计算结果如下：

$$\Pr[k_1 \mid 1] = 1 \quad \Pr[k_2 \mid 1] = 0 \quad \Pr[k_3 \mid 1] = 0$$

$$\Pr[k_1 \mid 2] = \frac{6}{7} \quad \Pr[k_2 \mid 2] = \frac{1}{7} \quad \Pr[k_3 \mid 2] = 0$$

$$\Pr[k_1 \mid 3] = 0 \quad \Pr[k_2 \mid 3] = \frac{3}{4} \quad \Pr[k_3 \mid 3] = \frac{1}{4}$$

接下来计算：

$$H(K \mid C) = \frac{1}{8} \times 0 + \frac{7}{16} \times 0.59 + \frac{1}{4} \times 0.81 + \frac{3}{16} \times 0 = 0.46$$

可见，两个结果是一致的。

定义 A.7　(**伪密钥**)设(P,C,K,E,D)是一个密码体制，明文串为 $x_1 x_2 \cdots x_n$。用一个密钥加密，得到的密文串为 $y_1 y_2 \cdots y_n$。考虑唯密文攻击，假设攻击者(Eve)有无限的计算资源并且知道明文是哪一种自然语言，如英语。一般来说，Eve 能排除某些密钥，但仍存在许多可能的密钥，在这其中只有一个密钥是正确的，而那些可能的但不正确的密钥称为伪密钥。

例 A.4　假设 Eve 得到密文“WNAJW”，这个密文是使用移位密码得到的。

容易知道，两个有意义的明文串，分别是“river”和“arena”，其对应的可能的加密密钥为 $F(=5)$和 $w(=22)$。这两个密钥，一个是正确的密钥，一个是伪密钥。

下面将推导伪密钥数量的期望下界。

首先，定义自然语言 L 的熵(每字母)的含义，记为 H_L。H_L 应该是对有意义的明文串中的每个字母平均信息的度量(注意，一个随机的字母串具有的熵是 $\log_2 26 \approx 4.70$)。作为 H_L 的“一阶”近似值，这里使用 $H(P)$来表示。L 是英语的时候，通过使用第 1 章中的概率分布表可得：

$$H(P) \approx 4.19$$

当然，语言中相继的字母不是统计独立的，相继字母的相关性减少了熵。例如，在英语中，字母“Q”后面总是跟着字母“U”。作为“二阶”近似值，在计算所有字母分组的概率分布的熵，然后除以 2。一般地，定义 P^n 为构成所有 n 字母分组的概率分布的随机变量。对语言的熵使用以下相关定义。

定义 A.8　假设 L 是自然语言，语言的熵定义为：

$$H_L = \lim_{n \to \infty} \frac{H(P^n)}{n}$$

自然语言 L 的冗余度(Redundacy)定义为：

$$R_L = 1 - \frac{H_L}{\log_2 |P|}$$

值得注意的是 H_L 中度量了自然语言 L 中每个字母的平均熵，一个随机的语言具有熵值为 $\log_2 |P|$。R_L 度量了“多余字母”的比例，记为冗余度。

在英语中，可以通过大量字母分组的列表和它们的频率给出 $H(P^2)$的估计，如

$H(P^2)/2 \approx 3.90$。

可以继续通过字母分组列表，得到对语言的熵值 H_L 的估计。事实上，各种实验已经得出了一个经验性的结果：$1.0 \leqslant H_L \leqslant 1.5$；也就是说，英语的平均信息内容大概是每字母1～1.5比特。

假定使用1.25作为 H_L 的估计值，可得冗余度为0.75。这意味着英语有75%的冗余度。（当然这不是说任意地从英语文本中去掉3/4的字母，还可以阅读；而是说对一个充分大的 n，可以找到一种编码方法，将英语原文压缩到原来长度的1/4。）

给定 K 和 P^n 的概率分布，C^n 是密文的 n 字母分组，可以定义 C^n 为代表密文 n 字母分组的随机变量。

给定 $y \in C^n$，定义：

$$K(y) = \{k \in K\text{：存在 } x \in P^n\text{，使得 } \Pr[x] > 0, e_k(x) = y\}$$

也就是说，$K(y)$ 是一个密钥的集合，在这些密钥下 y 是长为 n 的有意义的明文串的密文；或者说，$K(y)$ 是密文为 y 的可能密钥的集合。

如果 y 是被观察的密文串，那么伪密钥的个数是 $|K(y)|-1$，因为只有一个可能的密钥是正确的密钥。伪密钥的平均数目（在所有可能的长为 n 的密文串上）记为 $\bar{s}_n$，其值的计算如下：

$$\begin{aligned}\bar{s}_n &= \sum_{y \in C^n} \Pr[y](|K(y)| - 1) \\ &= \sum_{y \in C^n} \Pr[y]\,|K(y)| - \sum_{y \in C^n} \Pr[y] \\ &= \sum_{y \in C^n} \Pr[y]\,|K(y)| - 1\end{aligned}$$

根据定理A.8，有：

$$H(K \mid C^n) = H(K) + H(P^n) - H(C^n)$$

同时需要估计：

$$H(P^n) \approx nH_L = n(1 - R_L)\log_2 |P|$$

其中，n 充分大，当然：

$$H(C^n) \leqslant n\log_2 |C|$$

如果 $|C| = |P|$，密文空间与明文空间相等，则有：

$$H(K \mid C^n) \geqslant H(K) - nR_L \log_2 |P| \tag{A.1}$$

将 $H(K|C^n)$ 和伪密钥的个数 $\bar{s}_n$ 关联起来，计算如下：

$$\begin{aligned}H(K \mid C^n) &= \sum_{y \in C^n} \Pr[y] H(K \mid y) \\ &\leqslant \sum_{y \in C^n} \Pr[y] \log_2 |K(y)| \\ &= \log_2 \sum_{y \in C^n} \Pr[y]\,|K(y)| \\ &= \log_2(\bar{s}_n + 1)\end{aligned} \tag{A.2}$$

其中，对于函数 $f(x) = \log_2 x$ 用到了Jensen不等式，于是得到不等式：

$$H(K \mid C^n) \leqslant \log_2(\bar{s}_n + 1) \tag{A.3}$$

将式(A.1)和式(A.2)结合起来,可得:

$$\log_2(\bar{s}_n + 1) \geqslant H(K) - nR_L\log_2 |P|$$

考虑等概率选取密钥的情况(此时 $H(K)$ 取最大值),将得到下面更具体的结论。

定理 A.9　设(P,C,K,E,D)是一个密码体制,$|C| = |P|$且密钥是等概率选取的。设R_L表示明文的自然语言的冗余度,那么给定一个充分长的密文串,伪密钥的期望数满足:

$$\bar{s}_n \geqslant \frac{|K|}{|P|^{nR_L}} - 1$$

当 n 增加时,$\frac{|K|}{|P|^{nR_L}} - 1$ 以指数速度趋近 0;同时如果 n 很小,$H(P^n)/n$ 可能对 H_L的估计不够好,此时上面的估计可能不大精确。

定义 A.9　一个密码体制的唯一解距离,定义为使得伪密钥的期望数等于零的 n 的值,记作 n_0,即在给定足够的计算时间下,攻击者能唯一计算出密钥所需密文的平均量。

如果在定理 A.9 中令$\bar{s}_n = 0$,解出 n,可以得到唯一解距离的一个近似估计,即:

$$n_0 \approx \frac{\log_2 |K|}{R_L \log_2 |P|}$$

例 A.5　在代换密码体制中,$|P| = 26$,$|K| = 26$。如果取 $R_L = 0.75$,就得到唯一解距离的估计:

$$n_0 \approx 88.4/(0.75 \times 4.7) \approx 25$$

这表明给定的密文串的长至少是 25 时,通常解密结果才是唯一的。

附录 B　AES 实例

输入明文：

“00　11　22　33　44　55　66　77　88　99　aa　bb　cc　dd　ee　ff”

明文是 128 位，于是 $N_b=4$。明文先进行预处理，得到初始 State：

$$\begin{bmatrix} 00 & 44 & 88 & cc \\ 11 & 55 & 99 & dd \\ 22 & 66 & aa & ee \\ 33 & 77 & bb & ff \end{bmatrix}$$

输入密钥 k：“00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f”

密钥是 128 位，于是 $N_k=4$。类似地，密钥进行预处理：

$$\begin{bmatrix} 00 & 04 & 08 & 0c \\ 01 & 05 & 09 & 0d \\ 02 & 06 & 0a & 0e \\ 03 & 07 & 0b & 0f \end{bmatrix}$$

密钥 k 扩展成包含 $N_r+1=11$ 个子密钥的扩展密钥 $w[4,44]$：

$$\begin{bmatrix} 00 & 04 & 08 & 0c & d6 & d2 & da & d6 & b6 & 64 & be & 68 & b6 & d2 & 6c & 04 & 47 & 95 & f9 & fd & 3c & a9 & 50 & ad \\ 01 & 05 & 09 & 0d & aa & af & a6 & ab & 92 & 3d & 9b & 30 & ff & c2 & 59 & 69 & f7 & 35 & 6c & 05 & aa & 9f & f3 & f6 \\ 02 & 06 & 0a & 0e & 74 & 72 & 78 & 76 & cf & bd & c5 & b3 & 74 & c9 & 0c & bf & f7 & 3e & 32 & 8d & a3 & 9d & af & 22 \\ 03 & 07 & 0b & 0f & fd & fa & f1 & fe & 0b & f1 & 00 & fe & 4e & bf & bf & 41 & bc & 03 & bc & fd & e8 & eb & 57 & aa \end{bmatrix}$$

$$\begin{matrix} 5e & f7 & a7 & 0a & 14 & e3 & 44 & 4e & 47 & a4 & e0 & ae & 54 & f0 & 10 & be & 13 & e3 & f3 & 4d \\ 39 & a6 & 55 & a3 & f9 & 5f & 0a & a9 & 43 & 1c & 16 & bf & 99 & 85 & 93 & 2c & 11 & 94 & 07 & 2b \\ 0f & 92 & 3d & 1f & 70 & e2 & df & c0 & 87 & 65 & ba & 7a & 32 & 57 & ed & 97 & 1d & 4a & a7 & 30 \\ 7d & 96 & c1 & 6b & 1a & 8c & 4d & 26 & 35 & b9 & f4 & d2 & d1 & 68 & 9c & 4e & 7f & 17 & 8b & c5 \end{matrix}\Bigg]$$

首先是 AES 加密。

第 0 轮(初始轮)明文的初始 State 与 $w[0]$进行异或：

$$\text{AddRoundKey}(0)：\begin{bmatrix} 00 & 40 & 80 & c0 \\ 10 & 50 & 90 & d0 \\ 20 & 60 & a0 & e0 \\ 30 & 70 & b0 & f0 \end{bmatrix}$$

第 1 轮：

$$\text{SubBytes}(1)：\begin{bmatrix} 63 & 09 & cd & ba \\ ca & 53 & 60 & 70 \\ b7 & d0 & e0 & e1 \\ 04 & 51 & e7 & 8c \end{bmatrix} \qquad \text{ShiftRows}(1)：\begin{bmatrix} 63 & 09 & cd & ba \\ 53 & 60 & 70 & ca \\ e0 & e1 & b7 & d0 \\ 8c & 04 & 51 & e7 \end{bmatrix}$$

MixColumns(1)：$\begin{bmatrix} 5f & 57 & f7 & 1d \\ 72 & f5 & be & b9 \\ 64 & bc & 3b & f9 \\ 15 & 92 & 29 & 1a \end{bmatrix}$　AddRoundKey(1)：$\begin{bmatrix} 89 & 85 & 2d & cb \\ d8 & 5a & 18 & 12 \\ 10 & ce & 43 & 8f \\ e8 & 68 & d8 & e4 \end{bmatrix}$

第 2 轮：

SubBytes(2)：$\begin{bmatrix} a7 & 97 & d8 & 1f \\ 61 & be & ad & c9 \\ ca & 8b & 1a & 73 \\ 9b & 45 & 61 & 69 \end{bmatrix}$　ShiftRows(2)：$\begin{bmatrix} a7 & 97 & d8 & 1f \\ be & ad & c9 & 61 \\ 1a & 73 & ca & 8b \\ 69 & 9b & 45 & 61 \end{bmatrix}$

MixColumns(2)：$\begin{bmatrix} ff & 31 & 64 & 77 \\ 87 & d8 & 51 & 3a \\ 96 & 6a & 51 & d0 \\ 84 & 51 & fa & 09 \end{bmatrix}$　AddRoundKey(2)：$\begin{bmatrix} 49 & 55 & da & 1f \\ 15 & e5 & ca & 0a \\ 59 & d7 & 94 & 63 \\ 8f & a0 & fa & f7 \end{bmatrix}$

第 3 轮：

SubBytes(3)：$\begin{bmatrix} 3b & fc & 57 & c0 \\ 59 & d9 & 74 & 67 \\ cb & 0e & 22 & fb \\ 73 & e0 & 2d & 68 \end{bmatrix}$　ShiftRows(3)：$\begin{bmatrix} 3b & fc & 57 & c0 \\ d9 & 74 & 67 & 59 \\ 22 & fb & cb & 0e \\ 68 & 73 & e0 & 2d \end{bmatrix}$

MixColumns(3)：$\begin{bmatrix} 4c & f7 & 2c & 53 \\ 9c & 71 & 3f & 4d \\ 1e & f0 & 86 & f2 \\ 66 & 76 & 8e & 56 \end{bmatrix}$　AddRoundKey(3)：$\begin{bmatrix} fa & 25 & 40 & 57 \\ 63 & b3 & 66 & 24 \\ 6a & 39 & 8a & 4d \\ 28 & c9 & 31 & 17 \end{bmatrix}$

第 4 轮：

SubBytes(4)：$\begin{bmatrix} 2d & 3f & 09 & 5b \\ fb & 6d & 33 & 36 \\ 02 & 12 & 7e & e3 \\ 34 & dd & c7 & f0 \end{bmatrix}$　ShiftRows(4)：$\begin{bmatrix} 2d & 3f & 09 & 5b \\ 6d & 33 & 36 & fb \\ 7e & e3 & 02 & 12 \\ f0 & 34 & dd & c7 \end{bmatrix}$

MixColumns(4)：$\begin{bmatrix} 63 & fc & 97 & 75 \\ 85 & 53 & be & 47 \\ b7 & 8d & 47 & d6 \\ 9f & f9 & 8e & 91 \end{bmatrix}$　AddRoundKey(4)：$\begin{bmatrix} 24 & 69 & 6e & 88 \\ 72 & 66 & d2 & 42 \\ 40 & b3 & 75 & 5b \\ 23 & fa & 32 & 6c \end{bmatrix}$

第 5 轮：

SubBytes(5)：$\begin{bmatrix} 36 & f9 & 9f & c4 \\ 40 & 33 & b5 & 2c \\ 09 & 6d & 9d & 39 \\ 26 & 2d & 23 & 50 \end{bmatrix}$　ShiftRows(5)：$\begin{bmatrix} 36 & f9 & 9f & c4 \\ 33 & b5 & 2c & 40 \\ 9d & 39 & 09 & 6d \\ 50 & 26 & 2d & 23 \end{bmatrix}$

$$\text{MixColumns}(5): \begin{bmatrix} f4 & 32 & 75 & 1d \\ bc & e5 & f1 & d0 \\ d4 & 54 & d6 & 3b \\ 54 & d0 & c5 & 3c \end{bmatrix} \quad \text{AddRoundKey}(5): \begin{bmatrix} c8 & 9b & 25 & b0 \\ 16 & 7a & 02 & 26 \\ 77 & c9 & 79 & 19 \\ bc & 3b & 92 & 96 \end{bmatrix}$$

第 6 轮：

$$\text{SubBytes}(6): \begin{bmatrix} e8 & 14 & 3f & e7 \\ 47 & da & 77 & f7 \\ f5 & dd & b6 & d4 \\ 65 & e2 & 4f & 90 \end{bmatrix} \quad \text{ShiftRows}(6): \begin{bmatrix} e8 & 14 & 3f & e7 \\ da & 77 & f7 & 47 \\ b6 & d4 & f5 & dd \\ 90 & 65 & e2 & 4f \end{bmatrix}$$

$$\text{MixColumns}(6): \begin{bmatrix} 98 & 00 & 6b & 8e \\ 16 & f8 & 2c & 5a \\ ee & 7f & 04 & d0 \\ 74 & 55 & 9c & 36 \end{bmatrix} \quad \text{AddRoundKey}(6): \begin{bmatrix} c6 & f7 & cc & 84 \\ 2f & 5e & 79 & f9 \\ e1 & ed & 39 & cf \\ 09 & c3 & 5d & 5d \end{bmatrix}$$

第 7 轮：

$$\text{SubBytes}(7): \begin{bmatrix} b4 & 68 & 4b & 5f \\ 15 & 58 & b6 & 99 \\ f8 & 55 & 12 & 8a \\ 01 & 2e & 4c & 4c \end{bmatrix} \quad \text{ShiftRows}(7): \begin{bmatrix} b4 & 68 & 4b & 5f \\ 58 & b6 & 99 & 15 \\ 12 & 8a & f8 & 55 \\ 4c & 01 & 2e & 4c \end{bmatrix}$$

$$\text{MixColumns}(7): \begin{bmatrix} c5 & 9a & f0 & 98 \\ 7e & 9b & 5f & c6 \\ 1c & d2 & 4b & 34 \\ 15 & 86 & e0 & 39 \end{bmatrix} \quad \text{AddRoundKey}(7): \begin{bmatrix} d1 & 79 & b4 & d6 \\ 87 & c4 & 55 & 6f \\ 6c & 30 & 94 & f4 \\ 0f & 0a & ad & 1f \end{bmatrix}$$

第 8 轮：

$$\text{SubBytes}(8): \begin{bmatrix} 3e & b6 & 8d & f6 \\ 17 & 1c & fc & a8 \\ 50 & 04 & 22 & bf \\ 76 & 67 & 95 & c0 \end{bmatrix} \quad \text{ShiftRows}(8): \begin{bmatrix} 3e & b6 & 8d & f6 \\ 1c & fc & a8 & 17 \\ 22 & bf & 50 & 04 \\ c0 & 76 & 67 & 95 \end{bmatrix}$$

$$\text{MixColumns}(8): \begin{bmatrix} ba & a1 & d5 & 5f \\ a0 & f9 & 51 & 41 \\ 3d & b5 & 2c & 4d \\ e7 & 6e & ba & 23 \end{bmatrix} \quad \text{AddRoundKey}(8): \begin{bmatrix} fd & 05 & 35 & f1 \\ e3 & e5 & 47 & fe \\ ba & d0 & 96 & 37 \\ d2 & d7 & 4e & f1 \end{bmatrix}$$

第 9 轮：

$$\text{SubBytes}(9): \begin{bmatrix} 54 & 6b & 96 & a1 \\ 11 & d9 & a0 & bb \\ f4 & 70 & 90 & 9a \\ b5 & 0e & 2f & a1 \end{bmatrix} \quad \text{ShiftRows}(9): \begin{bmatrix} 54 & 6b & 96 & a1 \\ d9 & a0 & bb & 11 \\ 90 & 9a & f4 & 70 \\ a1 & b5 & 0e & 2f \end{bmatrix}$$

MixColumns(9)：$\begin{bmatrix} e9 & 02 & 1b & 35 \\ f7 & 30 & f2 & 3c \\ 4e & 20 & cc & 21 \\ ec & f6 & f2 & c7 \end{bmatrix}$　AddRoundKey(9)：$\begin{bmatrix} bd & f2 & 0b & 8b \\ 6e & b5 & 61 & 10 \\ 7c & 77 & 21 & b6 \\ 3d & 9e & 6e & 89 \end{bmatrix}$

第 10 轮(最后一轮)：

SubBytes(10)：$\begin{bmatrix} 7a & 89 & 2b & 3d \\ 9f & d5 & ef & ca \\ 10 & f5 & fd & 4e \\ 27 & 0b & 9f & a7 \end{bmatrix}$　ShiftRows(10)：$\begin{bmatrix} 7a & 89 & 2b & 3d \\ d5 & ef & ca & 9f \\ fd & 4e & 10 & f5 \\ a7 & 27 & 0b & 9f \end{bmatrix}$

AddRoundKey(10)：$\begin{bmatrix} 69 & 6a & d8 & 70 \\ c4 & 7b & cd & b4 \\ e0 & 04 & b7 & c5 \\ d8 & 30 & 80 & 5a \end{bmatrix}$

加密后的密文是：

“69　c4　e0　d8　6a　7b　04　30　d8　cd　b7　80　70　b4　c5　5a”

下面是 AES 的解密过程。

密文的初始 State：

$$\begin{bmatrix} 69 & 6a & d8 & 70 \\ c4 & 7b & cd & b4 \\ e0 & 04 & b7 & c5 \\ d8 & 30 & 80 & 5a \end{bmatrix}$$

第 0 轮：

AddRoundKey(0)：$\begin{bmatrix} 7a & 89 & 2b & 3d \\ d5 & ef & ca & 9f \\ fd & 4e & 10 & f5 \\ a7 & 27 & 0b & 9f \end{bmatrix}$

第 1 轮：

InvShiftRows(1)：$\begin{bmatrix} 7a & 89 & 2b & 3d \\ 9f & d5 & ef & ca \\ 10 & f5 & fd & 4e \\ 27 & 0b & 9f & a7 \end{bmatrix}$　InvSubBytes(1)：$\begin{bmatrix} bd & f2 & 0b & 8b \\ 6e & b5 & 61 & 10 \\ 7c & 77 & 21 & b6 \\ 3d & 9e & 6e & 89 \end{bmatrix}$

AddRoundKey(1)：$\begin{bmatrix} e9 & 02 & 1b & 35 \\ f7 & 30 & f2 & 3c \\ 4e & 20 & cc & 21 \\ ec & f6 & f2 & c7 \end{bmatrix}$　InvMixColumns(1)：$\begin{bmatrix} 54 & 6b & 96 & a1 \\ d9 & a0 & bb & 11 \\ 90 & 9a & f4 & 70 \\ a1 & b5 & 0e & 2f \end{bmatrix}$

第 2 轮：

$$\text{InvShiftRows(2)：}\begin{bmatrix}54 & 6b & 96 & a1\\ 11 & d9 & a0 & bb\\ f4 & 70 & 90 & 9a\\ b5 & 0e & 2f & a1\end{bmatrix} \quad \text{InvSubBytes(2)：}\begin{bmatrix}fd & 05 & 35 & f1\\ e3 & e5 & 47 & fe\\ ba & d0 & 96 & 37\\ d2 & d7 & 4e & f1\end{bmatrix}$$

$$\text{AddRoundKey(2)：}\begin{bmatrix}ba & a1 & d5 & 5f\\ a0 & f9 & 51 & 41\\ 3d & b5 & 2c & 4d\\ e7 & 6e & ba & 23\end{bmatrix} \quad \text{InvMixColumns(2)：}\begin{bmatrix}3e & b6 & 8d & f6\\ 1c & fc & a8 & 17\\ 22 & bf & 50 & 04\\ c0 & 76 & 67 & 95\end{bmatrix}$$

第 3 轮：

$$\text{InvShiftRows(3)：}\begin{bmatrix}3e & b6 & 8d & f6\\ 17 & 1c & fc & a8\\ 50 & 04 & 22 & bf\\ 76 & 67 & 95 & c0\end{bmatrix} \quad \text{InvSubBytes(3)：}\begin{bmatrix}d1 & 79 & b4 & d6\\ 87 & c4 & 55 & 6f\\ 6c & 30 & 94 & f4\\ 0f & 0a & ad & 1f\end{bmatrix}$$

$$\text{AddRoundKey(3)：}\begin{bmatrix}c5 & 9a & f0 & 98\\ 7e & 9b & 5f & c6\\ 1c & d2 & 4b & 34\\ 15 & 86 & e0 & 39\end{bmatrix} \quad \text{InvMixColumns(3)：}\begin{bmatrix}b4 & 68 & 4b & 5f\\ 58 & b6 & 99 & 15\\ 12 & 8a & f8 & 55\\ 4c & 01 & 2e & 4c\end{bmatrix}$$

第 4 轮：

$$\text{InvShiftRows(4)：}\begin{bmatrix}b4 & 68 & 4b & 5f\\ 15 & 58 & b6 & 99\\ f8 & 55 & 12 & 8a\\ 01 & 2e & 4c & 4c\end{bmatrix} \quad \text{InvSubBytes(4)：}\begin{bmatrix}c6 & f7 & cc & 84\\ 2f & 5e & 79 & f9\\ e1 & ed & 39 & cf\\ 09 & c3 & 5d & 5d\end{bmatrix}$$

$$\text{AddRoundKey(4)：}\begin{bmatrix}98 & 00 & 6b & 8e\\ 16 & f8 & 2c & 5a\\ ee & 7f & 04 & d0\\ 74 & 55 & 9c & 36\end{bmatrix} \quad \text{InvMixColumns(4)：}\begin{bmatrix}e8 & 14 & 3f & e7\\ da & 77 & f7 & 47\\ b6 & d4 & f5 & dd\\ 90 & 65 & e2 & 4f\end{bmatrix}$$

第 5 轮：

$$\text{InvShiftRows(5)：}\begin{bmatrix}e8 & 14 & 3f & e7\\ 47 & da & 77 & f7\\ f5 & dd & b6 & d4\\ 65 & e2 & 4f & 90\end{bmatrix} \quad \text{InvSubBytes(5)：}\begin{bmatrix}c8 & 9b & 25 & b0\\ 16 & 7a & 02 & 26\\ 77 & c9 & 79 & 19\\ bc & 3b & 92 & 96\end{bmatrix}$$

$$\text{AddRoundKey(5)：}\begin{bmatrix}f4 & 32 & 75 & 1d\\ bc & e5 & f1 & d0\\ d4 & 54 & d6 & 3b\\ 54 & d0 & c5 & 3c\end{bmatrix} \quad \text{InvMixColumns(5)：}\begin{bmatrix}36 & f9 & 9f & c4\\ 33 & b5 & 2c & 40\\ 9d & 39 & 09 & 6d\\ 50 & 26 & 2d & 23\end{bmatrix}$$

第 6 轮：

InvShiftRows(6)：$\begin{bmatrix} 36 & f9 & 9f & c4 \\ 40 & 33 & b5 & 2c \\ 09 & 6d & 9d & 39 \\ 26 & 2d & 23 & 50 \end{bmatrix}$　InvSubBytes(6)：$\begin{bmatrix} 24 & 69 & 6e & 88 \\ 72 & 66 & d2 & 42 \\ 40 & b3 & 75 & 5b \\ 23 & fa & 32 & 6c \end{bmatrix}$

AddRoundKey(6)：$\begin{bmatrix} 63 & fc & 97 & 75 \\ 85 & 53 & be & 47 \\ b7 & 8d & 47 & d6 \\ 9f & f9 & 8e & 91 \end{bmatrix}$　InvMixColumns(6)：$\begin{bmatrix} 2d & 3f & 09 & 5b \\ 6d & 33 & 36 & fb \\ 7e & e3 & 02 & 12 \\ f0 & 34 & dd & c7 \end{bmatrix}$

第 7 轮：

InvShiftRows(7)：$\begin{bmatrix} 2d & 3f & 09 & 5b \\ fb & 6d & 33 & 36 \\ 02 & 12 & 7e & e3 \\ 34 & dd & c7 & f0 \end{bmatrix}$　InvSubBytes(7)：$\begin{bmatrix} fa & 25 & 40 & 57 \\ 63 & b3 & 66 & 24 \\ 6a & 39 & 8a & 4d \\ 28 & c9 & 31 & 17 \end{bmatrix}$

AddRoundKey(7)：$\begin{bmatrix} 4c & f7 & 2c & 53 \\ 9c & 71 & 3f & 4d \\ 1e & f0 & 86 & f2 \\ 66 & 76 & 8e & 56 \end{bmatrix}$　InvMixColumns(7)：$\begin{bmatrix} 3b & fc & 57 & c0 \\ d9 & 74 & 67 & 59 \\ 22 & fb & cb & 0e \\ 68 & 73 & e0 & 2d \end{bmatrix}$

第 8 轮：

InvShiftRows(8)：$\begin{bmatrix} 3b & fc & 57 & c0 \\ 59 & d9 & 74 & 67 \\ cb & 0e & 22 & fb \\ 73 & e0 & 2d & 68 \end{bmatrix}$　InvSubBytes(8)：$\begin{bmatrix} 49 & 55 & da & 1f \\ 15 & e5 & ca & 0a \\ 59 & d7 & 94 & 63 \\ 8f & a0 & fa & f7 \end{bmatrix}$

AddRoundKey(8)：$\begin{bmatrix} ff & 31 & 64 & 77 \\ 87 & d8 & 51 & 3a \\ 96 & 6a & 51 & d0 \\ 84 & 51 & fa & 09 \end{bmatrix}$　InvMixColumns(8)：$\begin{bmatrix} a7 & 97 & d8 & 1f \\ be & ad & c9 & 61 \\ 1a & 73 & ca & 8b \\ 69 & 9b & 45 & 61 \end{bmatrix}$

第 9 轮：

InvShiftRows(9)：$\begin{bmatrix} a7 & 97 & d8 & 1f \\ 61 & be & ad & c9 \\ ca & 8b & 1a & 73 \\ 9b & 45 & 61 & 69 \end{bmatrix}$　InvSubBytes(9)：$\begin{bmatrix} 89 & 85 & 2d & cb \\ d8 & 5a & 18 & 12 \\ 10 & ce & 43 & 8f \\ e8 & 68 & d8 & e4 \end{bmatrix}$

AddRoundKey(9)：$\begin{bmatrix} 5f & 57 & f7 & 1d \\ 72 & f5 & be & b9 \\ 64 & bc & 3b & f9 \\ 15 & 92 & 29 & 1a \end{bmatrix}$　InvMixColumns(9)：$\begin{bmatrix} 63 & 09 & cd & ba \\ 53 & 60 & 70 & ca \\ e0 & e1 & b7 & d0 \\ 8c & 04 & 51 & e7 \end{bmatrix}$

第 10 轮(最后一轮):

InvSubBytes(10): $\begin{bmatrix} 63 & 09 & cd & ba \\ ca & 53 & 60 & 70 \\ b7 & d0 & e0 & e1 \\ 04 & 51 & e7 & 8c \end{bmatrix}$ InvShiftRows(10): $\begin{bmatrix} 00 & 40 & 80 & c0 \\ 10 & 50 & 90 & d0 \\ 20 & 60 & a0 & e0 \\ 30 & 70 & b0 & f0 \end{bmatrix}$

AddRoundKey(10): $\begin{bmatrix} 00 & 44 & 88 & cc \\ 11 & 55 & 99 & dd \\ 22 & 66 & aa & ee \\ 33 & 77 & bb & ff \end{bmatrix}$

解密后的明文是:

"00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff"

参 考 文 献

[1] Diffie W., Hellman M. E.. New Directions in Cryptography. IEEE Transaction on Information Theory, 1976, 22(6): 644～654.

[2] Shannon C. E.. Communication Theory of Secrecy Systems. Bell System Technical Journal, 1949, 28: 656～715.

[3] Douglas R. S.. Cryptography: Theory and Practice, 3rd Edition. Chapman & Hall/CRC, 2005.

[4] Buchmann J.. Introduction to Cryptography. Springer, 2004.

[5] Filiol E., Fontaine C.. A New Ultrafast Stream Ciphers Design: COS Ciphers. In: The 8th IMA Conference on Cryptography and Coding, LNCS, Springer, 2001, 2260: 85～98.

[6] Lamport L.. Password Authentication with Insecure Communication. Communications of the ACM, 1981, 24(11): 770～772.

[7] Tzeng W. G., Tzeng Z. J.. A Public-key Traitor Tracing Scheme with Revocation Using Dynamic Shares. PKC 2001. Springer, 2001, 207～224.

[8] Ghodosi H., Saeednia S.. Modification to Self-certified Group-oriented Cryptosystem, Without Combiner. Electronics Letters, 2001, 37(2): 86～87.

[9] Stallings W.,密码编码学与网络安全：原理与实践.二版.北京：电子工业出版社,2001.

[10] Needham R., Schroeder M.. Using Encryption for Authentication in Large Networks of Computers. Communications of the ACM, 1978, 21(12): 993～999.

[11] IEEE Std 1363-2000: IEEE Standard Specifications for Public-Key Cryptography. Institute of Electrical and Electronics Engineers, 2000.

[12] CAO Y. Y., FU C.. An Efficient Implementation of RSA Digital Signature Algorithm. Proceedings of the 2008 International Conference on Intelligent Computation Technology and Automation(ICICTA' 2008), IEEE Computer Society, Washington D. C., USA, 2008, 100～103.

[13] Wiener M. J.. Cryptanalysis of Short RSA Secret Exponents. IEEE Transaction on Information Theory, 1990, 36(3): 553～558.

[14] Frankel Y.. MacKenzie P. D., Yung M.. Robust Efficient Distributed RSA-key Generation. Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, 1998, 663～672.

[15] Koblitz N.. Elliptic Curve Cryptosystems. Mathematics of Computation, 1987, 48: 203～209.

[16] Schoof R.. Counting Points on Elliptic Curves Over Finite Fields. Theorie des Nombres de Bordeaux, 1995(7): 219～254.

[17] Schoof R.. Elliptic Curves over Finite Fields and the Computation of Square Roots Mod P, Math, Comp, 1985(44): 483～494.

[18] Atkin A. O. L.. Morain F.. Elliptic Curve and Primality Proving. Mathematics of Computation, 1993, 61(203): 29～68.

[19] Atkin, A. O. L.. The Number of Points on an Elliptic Curve Module a Prime. Draft, Available on http://listserv.nodak.edu/archives/nmbrthry.html, 1992.

[20] Elkies N. D.. Elliptic and Modular Curves over Finite Fields and Related Computational Issues. Computational Perspective on Number Theory. AMS/International Press, 1998, 21～76.

[21] Koblitz N.. The State of Elliptic Curve Cryptography. Designs Codes and Cryptography, 2000(19): 173～193.

[22] IEEE 1363a. IEEE Standard Specifications for Public-Key Cryptography-Amendment 1：Additional Techniques(S). Institute of Electrical and Electronics Engineers，2004.
[23] Benaissa M.，Lim W. M.. Design of flexible GF(2^m) Elliptic Curve Cryptography Processors. IEEE Transactions on Very Large Scale Integration (VLSI) Systems，2006，14(6)：659～662.
[24] Merkle C.，Hellman M E. Hiding Information and Signatures in Trapdoor Knapsacks. IEEE Transactions on Information Theory，1978，24(5)：525～530.
[25] Shamir A.. A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem. IEEE Transactions on Information Theory，1984，30(5)：699～704.
[26] IEEE 1363. 2. IEEE Standard for Specifications for Password-Based Public Key Cryptographic Techniques. Institute of Electrical and Electronics Engineers，2008.
[27] Blum M.，Goldwasser S.. An Efficient Probabilistic Public-key Encryption Scheme Which Hides All Partial Information. Advances in Cryptology-Crypto '84，Springer，1985. 289～299.
[28] Hoffstein J.，Pipher J.，Silverman J. H.. A Ring Based Public Key Cryptosystem. LNCS，1423. 267～288.
[29] McEliece R. J.. A Public-key Cryptosystem Based on Algebraic Coding Theory. JPL DSN Progress Report，1978. 114～116.
[30] Kazuo O.，Pei D. Y.. Cryptanalysis of the Original McEliece Cryptosystem. Advances in Cryptology-ASIACRYPT'98. LNCS，1514. 187～199.
[31] 王尚平，王育民，张亚玲. 基于 DSA 及 RSA 的证实数字签名方案. 软件学报，2003，14(3)：489～593
[32] Rivest R. L.，Shamir A，Adleman L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM，1978，21(2)：120～126.
[33] NIST FIPS PUB 186. Digital Signature Standard(S). Department of Commerce，NIST，1994.
[34] ELGAMAL T.. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. IEEE Transactions on Information Theory，1985，31(4)：469～472.
[35] ANSI X9. 62. Public Key Cryptography for the Financial Service Industry：The Elliptic Curve Digital Signature Algorithm (ECDSA)，1999.
[36] ANSI X9. 62. Public Key Cryptography for the Financial Service Industry：Elliptic Curve Key Agreement and Key Transport Protocols Working Draft，1999.
[37] Chaum D.，Antwerpen H. V.. Undeniable Signature Undeniable Signature. Workshop on the Theory and Application of Cryptographic Techniques，1989(434)：212～216.
[38] Håstad J.. On Using RSA with Low Exponent in a Public Key Network，Advances in Cryptology-CRYPTO'85. LNCS，218：403～408.
[39] Stadler M.，Piveteau J. M.，Camenisch J. Fair Blind Signatures. LNCS，1995，921：209～219.
[40] Heyst E. V.，Pedersen T. P.. How to Make Efficient Fail-stop Signatures. LNCS，1992(658)：366～377.
[41] Yu Y.，Xu C. X.. An Efficient Anonymous Proxy Signature Scheme with Provable Security. Computer Standards&Interfaces，2009(31)：348～353.
[42] Bellare M.，Rogaway P.. Random Oracles Are Practical：A paradigm for Designing Efficient Protocols. Proceedings of the first ACM Conference on Computer and Communications Security. Fairfax，USA，1993. 62～73.
[43] 林品，吴文玲，武传坤. 一种基于分组密码的 hash 函数. 软件学报，2009，20(3)：682～691.
[44] Damgard I. B.. A Design Principle for Hash Functions. CRYPTO 1989. LNCS 435，1990. 416～427.
[45] Rivest R.. The MD5 Message-digest Algorithm. Internet Activity Board. Internet Privacy Task Force，1992.
[46] NIST FIPS 180-1. Secure Hash Standard (SHS). Federal Information Processing Standards

Publication,1995.

[47] NIST FIPS 198. The Keyed-Hash Message Authentication Code (HMAC). Federal Information Processing Standards Publication,2002.

[48] 王大印,林东岱,吴文玲.一种可并行的消息认证码.软件学报,2007,18(7):1756～1764.

[49] 陈杰,胡予濮,韦永壮,等.随机消息伪造攻击 PMAC 和 TMAC-V.计算机学报,2007,30(10):1827～1832.

[50] Bellare M.,Canetti R.,Krawczyk H.. Keying Hash Functions for Message Authentication. Advances in Cryptography,Crypto96 Proceeding,1996.1～19.

[51] Toyran M.,Berber S.. Efficient Implementation of Diffie-Hellman (DH) Key Distribution Algorithm in Pool-based Cryptographic Systems. Electrical and Electronics Engineering (ELECO) 2009. International Conference on Browse Conference Publications,II:271～275.

[52] Schneier B.. Applied Cryptography: Protocols,Algorithms and Source Code in C,Second Edition. John Wiley & Sons. Inc.,1996.

[53] Stallings W.. Cryptography and Network Security Principles and Practices. New Jersey: Prentice Hall,2005.

[54] Menezes A. J.,Orschot P. V.. Vanstone S. Handbook of Applied Cryptography. CRC Press,1996.

[55] 温蜜,陈克非,郑燕飞.传感器网络中一种可靠的对密钥更新方案.软件学报,2007,18(5):1232～1245.

[56] Blom R.. An Optimal Class of Symmetric Key Generation Systems,Proceedings of the EUROCRYPT 84 Workshop on Advances in Cryptology: Theory and Application of Cryptographic Techniques,1985,335～338.

[57] Blundo C.,Santis A. D.,Herzberg A.. Perfectly-Secure Key Distribution for Dynamic Conferences. Proceedings of the 12th Annual International Conference on Advances in Cryptology,1993.471～486.

[58] 刘克龙,卿斯汉,蒙杨.一种利用公钥体制改进 Kerberos 协议的方法.软件学报,2001,12(6):872～877.

[59] 文铁华,谷士文.增强 Kerberos 协议安全性的改进方案.通信学报,2004,25(6):76～79

[60] RFC 4120. The Kerberos Network Authentication Service (V5). IETF,2005.

[61] Kohl J.,Neuman B. C.,Ts'o T. Y.. The Evolution of the Kerberos Authentication Service. Distributed Open Systems. IEEE Computer Society Press,1994.78～94.

[62] Brazier F.,Johansen D.. Distributed Open Systems. IEEE Computer Society Press,1994.78～94

[63] Needham R. M.,Schroeder M. D.. Using Encryption for Authentication in Large Networks of Computers. Communications of the ACM,1978,21(12):993～999.

[64] Perlman R.. An Overview of PKI Trust Models. IEEE Network,1999,13(6):38～43.

[65] Hourley R.. Internet X.509 Public Key Infrastructure Certificate and CRL Profile,RFC 2459,Jan. 1999.

[66] RFC 5280. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. IETF,2008.

[67] Belgers W.. UNIX Password Security. http://www.ja.net/CERT/Belgers/UNIX-password-security.html,1993.

[68] Morris R.,Thompson K.. Password Security: A Case History. Communications of the ACM,1979,22(11):594～597.

[69] Feldmeier D. C.,Karn P. R.. UNIX Password Security-Ten Years Later. Proceedings of Advances in Cryptology-CRYPTO '89,1989.

[70] Klein D. V.. Foiling the Cracker: A Survey of,and Improvements to,Password Security. Proceedings of the 2nd USENIX Security Workshop,1990.

[71] FIPS 46-3. Data Encryption Standard. Federal Information Processing Standard. National Bureau of Standards,1999.

[72] FIPS 197. Advanced Encryption Standard. Federal Information Processing Standard. National Bureau of Standards,2001.

[73] Grampp F. T. ,Morris R. H. . Unix Operating System Security. AT&T Bell Labs Technical Journal, 1984,63(8): 1649～1672.

[74] Huang Y. , Rine D. , Wang X. H. . A JCA-based Implementation Framework for Threshold Cryptography. Computer Security Applications Conference,2001. 85～91.

[75] Sun Microsystems Inc. Java Cryptography Architecture API Specification & Reference. http://java.sun.com/j2se/sdk/1.3/docs/guide/security/CryptoSpec.html,1999.

[76] Zhang W. B. ,Huang X. ,Zhang B. . An Aspect-Oriented Modeling Approach to Predict Performance of JCA-Based Systems. Proceedings of Interoperability for Enterprise Software and Applications, 2009. 140～146.

[77] Sun Microsystems Inc. Java 2 Platform, Enterprise Edition Connector Architecture Specification. http://java.sun.com.

[78] Bisel L. D. . The Role of SSL in Cyber Security. IT Professional,2007,9(2): 22～25.

[79] Zhao H. W. , Liu R. X. . A Scheme to Improve Security of SSL. Circuits, Communications and Systems,2009. 401～404.

[80] Hickman E. B. . The SSL Protocol. Netscape Communication Corp,1995.

[81] Karjoth G. . An Operational Semantics of Java 2 Access Control. Proceedings Computer Security Foundations Workshop,2000. 224～232.

[82] Lai C. ,Gong L. . User Authentication and Authorization in the Java™ platform. Computer Security Applications Conference,1999. 285～290.

[83] Pinagapani S. ,Xu D. X. ,Kong J. . A Comparative Study of Access Control Languages. Third IEEE International Conference on Secure Software Integration and Reliability Improvement, 2009. 407～412.

[84] Sun Microsystems Inc. JAAS Reference Guide. http://java.sun.com/javase/6/docs/ technotes/guides/security/jaas/ JAASRefGuide.html.